AFRL-RI-RS-TR-2011-007

ENTERPRISE INFORMATION LIFECYCLE MANAGEMENT

BBN TECHNOLOGIES CORPORATION

JANUARY 2011

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■**UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2011-007 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

JAMES P. HANNA
Work Unit Manager

/s/

JULIE BRICHACEK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* January 2011	**2. REPORT TYPE** Final Technical Report	**3. DATES COVERED** *(From - To)* October 2009 – October 2010

4. TITLE AND SUBTITLE ENTERPRISE INFORMATION LIFECYCLE MANAGEMENT	**5a. CONTRACT NUMBER** FA8750-10-C-0021
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 62788F
6. AUTHOR(S) Joseph Loyall, Jeffrey Cleveland, Jonathan Webb, Shane Clark	**5d. PROJECT NUMBER** S2LM
	5e. TASK NUMBER 05
	5f. WORK UNIT NUMBER 06
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) BBN Technologies Corporation, 10 Moulton Street, Cambridge MA 02138-1119	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RISE, 525 Brooks Road, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RISE
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2011-007

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA#88ABW-2011-0025
Date Cleared: 5 January 2011.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Information Lifecycle Management is a key aspect of Information Management, determining when information should be moved to backing store to free up local, high speed storage for critical information needs. The Value Factor based Information Lifecycle Management (VFILM) project under the Enterprise Information Lifecycle Management contract developed concepts and a prototype for mission-driven information lifecycle management that includes automated triggering of information migration based on mission events and mission-based policy, valuation of information using dependencies, and migration and retrieval of information objects and groups. The resulting prototype software works with AFRL Information Management services and repositories and has been demonstrated on relevant USAF operational scenarios. The software prototype was validated with experiments that evaluated its functionality, performance, and compared it to age-based Hierarchical Storage Management (HSM) approaches. This report is the final technical report for the VFILM project and describes the research, development, and evaluation results from the project.

15. SUBJECT TERMS
Information Lifecycle Management, Information Management, Fuzzy Logic, Hierarchical Storage Management

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** JAMES HANNA
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	105	**19b. TELEPHONE NUMBER** *(Include area code)* N/A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 SUMMARY

This document is the final technical report for the *Value Factor based Information Lifecycle Management (VFILM)* project under the Enterprise Information Lifecycle Management contract.

1.1 Goals of the VFILM Project

The goals of the VFILM project are to research, develop, and evaluate technology for managing the lifecycle of information and enable the use of Hierarchical Storage Management (HSM) in information-centric, mission-critical systems.

1.2 Summary of Major Results

VFILM developed concepts and a prototype for mission-driven information lifecycle management that includes automated triggering of information migration between storage levels based on mission events and mission-based policy, valuation of information based on its urgency to ongoing mission operations, grouping of information based on common attributes and dependencies, and migration and retrieval of information objects and groups.

The resulting prototype software works with AFRL Information Management (IM) services and repositories and was demonstrated on relevant USAF operational scenarios. We validated the software prototype with experiments that evaluated its functionality, performance, and compared it to age-based HSM approaches.

The VFILM project resulted in the following significant results:

- A prototype Information Lifecycle Management (ILM) service and HSM interface that provides mission-aware information valuation, mission-driven movement of information between levels of storage, and support for AFRL Phoenix IM services, Information Objects (IOs), and repositories.
- A novel approach to information valuation, supporting an extensible multi-factor assessment of the relative values of information using fuzzy logic. The approach produces a partial order of information depreciation, handles dynamic conditions that can change the worth of information, and avoids the thrashing that is possible with fixed or static valuation thresholds.
- A set of experimentation results and unit tests, which are useful as a functional and performance test suite for ILM services.

2.0 INTRODUCTION

2.1 Project Objective

The objective of the VFILM project is to research, develop, and evaluate technology for managing the lifecycle of information and enable the use of Hierarchical Storage Management (HSM) in information-centric, mission-critical systems.

VFILM developed concepts and a prototype for mission-driven information lifecycle management that includes automated triggering of information migration based on mission events and mission-based policy, valuation of information based on its urgency to ongoing mission operations, grouping of information based on common attributes and dependencies, and migration and retrieval of information objects and groups.

The resulting prototype software works with AFRL IM services and repositories and was demonstrated on relevant USAF operational scenarios. We validated the software prototype with experiments that evaluated its functionality, performance, and compared it to age-based HSM approaches.

2.2 Project Overview

The *VFILM* project ran from October 28, 2009 to October 28, 2010. BBN Technologies performed all technical work under the project for the project. The AFRL Program Manager (PM) was James Hanna. The BBN Principal Investigator (PI) and PM was Dr. Joseph Loyall. The BBN Technical Lead was Jonathan Webb. The primary BBN technical contributor was Jeffrey Cleveland.

2.3 Background

2.3.1 Information Lifecycle and Hierarchical Storage Management Systems

Information Lifecycle Management (ILM) solutions available at the beginning of this project took the following forms [4][25]:

- Storage-centric offerings, i.e., multiple storage solutions with different capacity and price characteristics and software and consulting to use it (the point of view of storage vendors);
- Technologies for *Hierarchical Storage Management (HSM)*, i.e., automatically moving information between storage levels (the point of view of some software vendors); or
- Business processes pertaining to the value, retention, and management of information (the point of view of some services companies).

Many current ILM and HSM technologies are variations on backup and retrieval software. They only work on files or documents; they move data based on age or time/frequency of access; and they are only triggered by storage full situations or by time.

Whereas much focus in ILM centers around the HSM part, most existing HSM offerings are mechanistic in nature, providing information movement and retrieval based on file systems and standardized control interfaces. They are invoked, manually or automatically, when storage space gets tight or based exclusively on time.

2.3.2 Hierarchical Storage Levels

It is common in HSM to categorize levels of storage (level 0, level 1, level 2, etc.) based on their relative speed, capacity, and cost, as shown in Figure 1. In general, lower storage levels are considered higher speed, lower latency, more costly, and lower capacity, as shown in Figure 2. In operational terms, level 0 is the most accessible to ongoing missions (e.g., onboard storage on tactical platforms), where levels 1 and higher increase in capacity and latency to access information by edge platforms.

In reality, these levels and the media that occupy them are not a total order. For example, the capacity of modern disk drives exceed the capacity of a single optical disk, the cost of tape is not necessarily less than that of optical disks, and the capacity of multiple optical disks and multiple tapes is comparable (virtually unlimited). Furthermore, some of the storage media are not very relevant. Specifically, it is not relevant to consider RAM and cache when discussing HSM technology, since they typically fall under the control of the operating system.

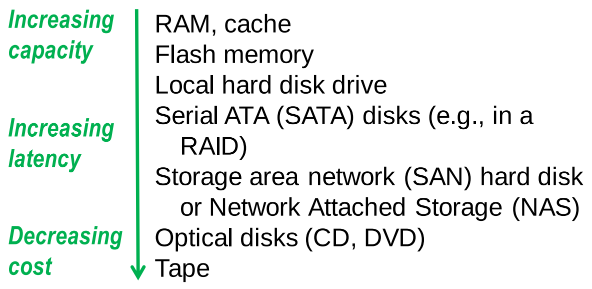


Figure 1. Example levels of hierarchical storage.

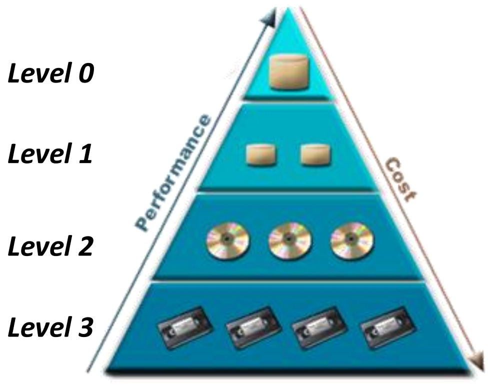


Figure 2. General characterization of hierarchical storage levels based on performance and cost.

2.4 Novel Research Aspects of the VFILM Project

The VFILM project addresses several novel aspects beyond the scope of existing HSM systems.

- The urgency of information is related to its importance to ongoing and future mission operations, whereas off-the-shelf HSM solutions focus on the simple characteristics of *age* and *time of last reference*. Some very high value information to particular missions can be referenced infrequently, e.g., nuclear command codes, emergency medical information, or panic room or alarm codes.
- Furthermore, the value of information is determined by varied and sometimes complex characteristics such as its source, type, relation to other information, and content. Ultimately the lifecycle of information depends on its semantics (how it is used) and connection to other information. Some information degrades in value when a mission epoch is reached, e.g., ISR information is less useful for real-time ISR at the end of a mission. Other information degrades when a new value is received, e.g., a new Blue Force Track supersedes old tracks reporting the position of the same forces. Frequently, information simply moves from being important for one use (e.g., an on-going mission) to being im-

portant for another (e.g., an after action review or mission reconstruction). Therefore, the relative importance of missions or use, and the affiliation of information to missions or uses of different importance should be considered in information lifecycle valuation.

- Whereas quantifying the age and time of last reference is straightforward, it is more difficult to quantify the semantics of its use, affiliation to important missions, relation to other sets of information, supersession by other values, and varied other characteristics. In this way, as part of VFILM, we have explored the research areas of content- and context-driven value judgments.

2.5 Primary results

The VFILM project resulted in the following significant results:

- A prototype ILM service and HSM interface that provides mission-aware information valuation, mission-driven movement of information between levels of storage, and support for AFRL Phoenix IM services, Information Objects, and repositories.
- A novel approach to information valuation, defining a *Value Depreciation Function (VDF)* that supports an extensible multi-factor assessment of the relative values of information using fuzzy logic. The VDF produces a partial order of information depreciation, handles dynamic conditions that can change the worth of information, and avoids the thrashing that is possible with fixed or static valuation thresholds.
- A set of experimentation results and unit tests, which are useful as a functional and performance test suite for ILM services.

2.6 Report organization

The report is organized as follows:

- Section 1.0 provides a one-page summary of the objectives and main results of the project.
- Section 2.0 provides an introduction, including the project objective, major results, background, research aspects, and the primary results.
- Section 3.0 provides the project's methods, assumptions, and procedures.
- Section 4.0 provides the project's results and discussion.
- Section 5.0 provides some concluding remarks.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

We used a spiral approach to rapidly prototype the VFILM capability. During the 12 month period of performance for the project, we developed two prototypes, each at the end of a six month spiral. The second, and final prototype, was an enhancement of and built upon the capabilities of the first.

This section describes the underlying assumptions and methods upon which we based the VFILM research, and the procedures that we followed in producing the VFILM results.

First, we focused on developing ILM capabilities for IM services and, as such, assumed the existence of a set of IM services. As described in Section 3.1, we focused on the *Phoenix* set of core IM services developed by AFRL under other projects. Section 3.2 describes the underlying set of requirement and technology drivers for establishing information lifecycle management in IM services as represented by Phoenix, i.e., the rationale, reasons, and needs motivating the development of ILM capabilities for IM services.

Section 3.3 describes the research challenges associated with designing and developing ILM capabilities. Section 3.4 describes the approach that we took to meeting these challenges. Some of the driving principles and assumptions underlying our approach described in Section 3.4 are that we should *reuse* as much of the existing Phoenix services as possible and, therefore, not change the semantics of Phoenix IM service operation needlessly. Sections 3.5 and 3.6 provide background information about HSM capabilities and the fuzzy logic basis for our valuation algorithm, respectively. Finally, Section 3.7 describes our experimentation methodology, including the experiments we defined and the metrics we collected.

3.1 Core Information Management Services

Figure 3 shows a set of core information management services for net-centric operations in AFRL's Phoenix software [9], which includes the following:

1. *Submission Service* – Receiving information objects entering the system as the result of publishing.

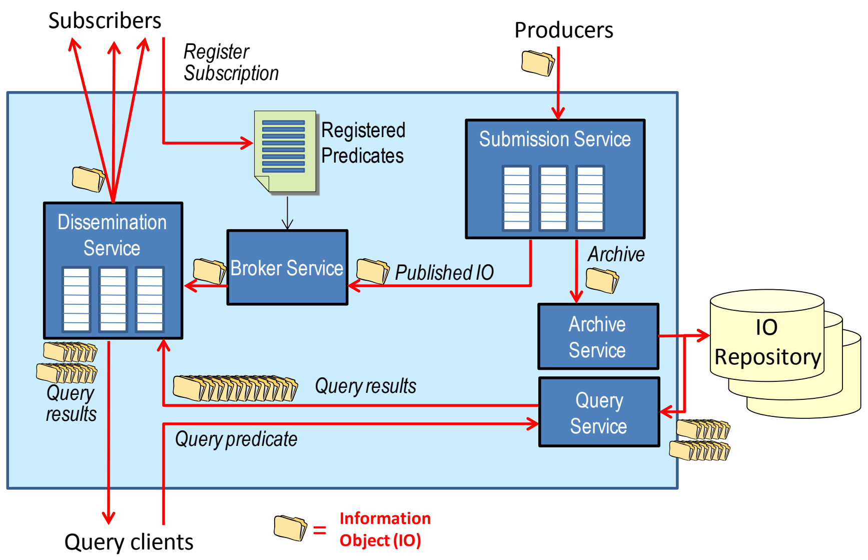


Figure 3. Core Information Management Services

2. *Brokering Service* – Matching of registered subscriptions with published information.
3. *Archiving Service* – Insertion of published information into an information repository.
4. *Query Service* –Evaluation of a query operation and subsequent retrieval of results.
5. *Dissemination Service* – Delivery of the results of brokering (a single IO) to matched clients (potentially many) and delivery of the results of a query (potentially many IOs) to the requestor (a single querying client).

3.2 The Need for Information Lifecycle Management

USAF AFRL IM systems and services need to include an ILM solution. Currently such systems, exemplified by the Phoenix Core Services, default to retaining all archived information. They provide no specific support for cleaning up information repositories that are reaching their saturation point, except through manual administrative interfaces and database interfaces outside the IM services. Thus, when repositories fill, information will be lost (in an unmanaged manner), archive operations will fail, software exceptions will be thrown, or in the worst case the IM services will fail.

Repositories will fill up and are likely to when they are needed the most, even with modern disks with the capacity of many Gigabytes or Terabytes. Consider that during Operation Anaconda in March 2002, U.S. air forces flew 65 combat sorties per day [13]. Thirty minutes of ISR video from a UAV in compressed MPEG-2 format requires 1.2 GB of disk space. A single high resolution RGB image in TIFF format (2248x2080 pixels) such as might be used for battle damage assessment or aimpoint generation requires over 13 MB of space.

An ILM service should manage how and what gets retained in each storage level, so that

- Critical information urgent to ongoing and upcoming missions is readily accessible.
- High speed, high cost storage is used for the information that is most critical to ongoing and upcoming missions.
- Movement of information is based on a decrease in value to ongoing or upcoming missions and storage space being needed for higher value, more critical information.
- Support for information repositories and IM operations, e.g., query and archive, is maintained.

In contrast to most of the solutions offered today, the military needs ILM solutions that are *mission-driven*, not simply triggered by a lack of available space or time, are *mission-aware*, not simply moving the oldest or least recently used data, and work with a variety of *structured information objects*, not just files or opaque documents.

3.3 Challenges of Information Lifecycle Management Design and Development

VFILM set out to tackle harder issues in information valuation, lifecycle management, and migration than traditional HSM solutions.

We set a design goal of separating the ILM service from the HSM functionality. The ILM determines the valuation of information and when information should be moved, and the HSM performs the actual movement and potentially monitoring of the storage.

Designing and developing an ILM service included the following challenges:

- Determining when information's value is sufficiently degraded (relative to other information) to move to backing store. We wanted an information valuation function that could consider a variety of measurable attributes and factors that could make information more or less valuable.

- Grouping related information that should be moved together. In IM systems like those we are targeting, information can be related through derivation or common association. For example, there might be multiple versions of information derived from a common raw sensor collection for different uses. Likewise, a set of information collected by a specific platform during a period of time or when the platform was in a particular area is useful for some purposes as a group. HSM systems that treat each information element as a separate, independent file lose the important associations that can affect its valuation and its use. We wanted our ILM service to be able to treat groups of information collectively, have group association factored into valuation and movement decisions, and recognize that groups can overlap.
- Triggering information movement at the appropriate times, i.e., by events associated with the mission profile and not simply when storage is exhausted or on a fixed schedule. Certainly having a fixed threshold and moving information when it becomes older than a certain value or has not been accessed in a pre-determined amount of time is easy to implement and simple to understand. However, this could lead to moving away important information, thrashing, and accidental memory overload or underutilization. We wanted the ILM to respond to multiple types of events that could independently trigger information valuation and/or movement, so that changes in mission conditions, patterns of usage, storage needs, policy, and other factors could trigger information valuation and movement.

Designing and developing an HSM interface included the following challenges:

- Designing or adapting an HSM that works with higher level concepts, including mission- and information-orientation. Most HSMs work at the filesystem level and do not consider what information is used for and its granularity. Furthermore, many HSMs are tied to particular filesystems, hardware, or processes, which does not provide the flexibility and power that military operations require.
- Making the HSM as invisible and automatic as possible. The HSM should be able to access and retrieve information no matter where it exists, and should retain the existing IM publish, archive, and query services. However, we realized also that supporting multiple repositories in hierarchical storage means that the query service can provide more options to query clients, and we wanted to expose these so that they could be used, while retaining the current IM query semantics as the default.
- Because of the emphasis on filesystems and processes, existing HSM solutions are applicable to enterprise situations. We recognized a large gap in using these solutions in tactical environments. Therefore, we wanted to build our ILM service and HSM interface to work with varying HSM solutions including those that would be appropriate for tactical situations and might not exist in a mature form at this time.

3.4 VFILM Approach

Our approach was to separate the ILM and HSM functionality and design and prototype them as separate services and mechanisms. Our rationale for doing this was twofold:

- It allows information valuation and movement to be treated as separate actions, but to be related if necessary. Information valuation changes in response to events associated with missions, information, or other elements. For example, when a mission ends, all information created or associated with that mission might depreciate in value, but there is no need to move the information at that point unless space is needed. Specifically, information valuation is triggered by events associated with changing the urgency or usefulness of information, whereas information movement is triggered by events associated with space usage and needs. Some events, such as an event associated with preparation for a mission, might trigger both information valuation and movement.

- Both elements are useful independently of, or in conjunction with, each other. An ILM service is a useful capability, and should be able to work with a variety of HSM functions. The ILM and the HSM should not be tightly coupled so that they can be used in other contexts and with other services/components.

The VFILM approach treats the ILM and HSM as *management services*. The following sections describe some of the foundations of the approach we took during the VFILM project to design and implement an ILM service and HSM capability.

3.4.1 Spiral Approach

The VFILM project was organized into two spirals, each approximately six months in duration. The first spiral designed and developed a core set of ILM and HSM functionality, producing a rapid prototype by midway through the project's period of performance and focusing on the following aspects:

- A first version ILM service
- A first version information valuation function
- A first set of ILM, system, and mission events
- An ILM-HSM interface and representative HSM capability
- An experimentation and evaluation plan

The second spiral then expanded upon and enhanced the Spiral One basis, producing the final prototype, and focusing on the following aspects:

- An enhanced ILM service
- A design and prototype implementation of ILM policy
- A design and implementation of information grouping
- Experimentation
- Demonstration of the VFILM prototype to AFRL
- Documentation of the VFILM prototype and research results.

3.4.2 Consistency with Existing Phoenix IM Services

One of the goals of our approach was to develop a service-oriented ILM, so that it will work with multiple service-oriented data archive services. We designed our prototype implementation to work with AFRL's Phoenix IM services and to minimize changes to the baseline Phoenix code.

3.4.3 Approach to ILM Service

Our approach was to create an *ILM service* that decides when information should move between levels, what information should move, and manages the following aspects of the information lifecycle:

- *The value of information* and the factors that can contribute to an increase or decrease in information value. Most ILM and HSM approaches available rely on recentness or frequency of information access to determine where it should reside [2]. This is insufficient in military situations because it ignores the inherent *value* of information and its *criticality* to ongoing operations. The access patterns of an IO contribute to its value, but are not the only factors. Mission-related information can be of higher or lower value based on its *urgency* to a mission and *how rapidly* it must be provided if accessed (necessitating its availability in local store). Other work has introduced *information value* as a key basis for ILM [2]. We expand on this work in two novel ways, by incorporating *mission-oriented* aspects that affect information value and by the way we represent the *value function*.
- *The grouping of information*, so that related information can be moved and/or retrieved together when appropriate. Traditional ILM and HSM focus on files and documents, an 80% solution appropriate for most business needs. The VFILM Approach supports files and documents, but also supports a richer organization of information, including instances of Phoenix Information classes. Our approach also supports grouping related and derivative information, e.g., based on association with a mission, sortie, or platform; common attributes (e.g., age); and other aspects.
- *The triggers of information movement*, which for existing HSM solutions are generally limited to when space becomes needed or specific scheduled times. This is not sufficient for USAF needs because when space is reaching capacity might be exactly when additional space is most needed by the mission and might be when the time and resources needed to move information to free up space are least available. We utilize a rich set of triggers to invoke the ILM to decide whether information should be moved, how much, and when, including the need for space, events such as mission epochs, and proactively when the ILM and HSM functions would not impact mission operations.

3.4.4 Approach to HSM Functionality – An Abstraction Layer

We chose to develop an interface to HSM functionality, i.e., the *ILM-HSM Adapter,* and to develop representative HSM functionality for the following reasons:

- We conducted an investigation of off-the-shelf HSM systems (described in Section 3.5). They varied significantly. Creating an abstraction layer and interface enabled us to design and implement the ILM service to work with a variety of HSM approaches.
- Our investigation indicated that utilizing an off-the-shelf HSM capability could entail significant investment of money and time, and would result in a VFILM prototype that would be tied to a specific filesystem or operating system.

3.5 Background on HSM Capabilities

Hierarchical Storage Managers come in many flavors and are described by various names, including Automated Availability, Data Migration, and Data Storage Management. Almost without exception, they deal with file-level migration.

High-end HSM implementations include IBM's Tivoli Storage Manager HSM, HP's File Archiving and Information Management Software, Unylogix HSM, and SGI's Data Migration Facility. Many of the high-end HSM implementations, such as IBM's Tivoli and HP's Information Management Software, offer integration with higher-level storage applications, such as Oracle databases or Exchange email servers. However, these integration options essentially map the application-specific stores to normal files and usually have fairly simplistic operational use.

For example, IBM's Tivoli Storage Manager for Databases provides integration with Oracle databases, but the only things that become managed by the HSM are the backups and archive files produced by the Oracle Recovery Manager (RMAN).

Many HSM systems strive to be invisible to users. These HSM systems, which include *IBM's Tivoli HSM for Windows*, *HP File Archiving*, *Unylogix HSM*, and *SGI's Data Migration Facility*, provide an interface that looks like a normal file system, but the files might be in any level of storage.

The high end HSM solutions were too expensive and proprietary for us to directly acquire and utilize for VFILM. For our VFILM prototype, we needed an HSM that is representative, but also economical. We investigated several open-source HSM solutions, including *DVD-Vault*, *OpenSMS*, *Sun's Storage and Archive Manager for the Quick File System (SAM/QFS)*, *OpenSolaris Automatic Data Migration (ADM)*, and *Sun's Lustre* file system.

From our investigation, we could not identify any suitable open-source HSM capability. Most work with specific filesystems. For example, Unylogix HSM supports only Solaris, while IBM Tivoli Storage Manager HSM and HP File Archiving support Windows. SGI's Data Migration Facility supports SUSE Linux Enterprise Server. Other HSMs are tied to specific hardware. For example, Unylogix HSM has a specific list of supported hardware storage devices (including optical jukeboxes and tape libraries). DVD-Vault works with DVD, Sony's ProData (PDD), or Blu-Ray SCSI Library. Some of the solutions worked with specific databases. HP Database Archive supports Microsoft SQL-Server and Oracle and Tivoli Storage Manager for Databases supports Oracle.

We concluded that the open-source HSM implementations were risky. They either assume the presence of specific hardware (e.g., DVD-Vault [6]), have not been maintained for several years (e.g., OpenSMS [21]), or are only partially open-source (e.g., SAM/QFS [22], ADM [23], and a beta version of an HSM component of the Lustre filesystem [18]).

As the most promising of our original investigation, we conducted a more in-depth investigation of the Lustre HSM component. Lustre is a high performance distributed filesystem, targeted for high performance computing clusters [5]. In the Lustre HSM project, Lustre was adding an interface to support multiple HSMs. At the time of our investigation, the Lustre HSM project had not yet released any software. The target system they were developing was not an HSM, but an interface for Lustre to enable it to interface to existing HSMs. It utilized the open source Robin Hood policy engine to monitor disk space usage and control the HSM functions. Because it was not targeting development of an HSM and because there was no released software, Lustre and the Lustre HSM project did not offer an off-the-shelf HSM for us to utilize.

Because of that, we developed representative HSM functionality for the VFILM prototype and demonstrations. We also developed an ILM-HSM adapter layer that works with our representative HSM functionality for demonstration and validation, and also serves as the interface point to off-the-shelf HSM capabilities.

3.6 Background in Fuzzy Logic

One of the key challenges for VFILM was capturing programmatically when an IO's value was sufficiently depreciated to warrant moving it from level 0 store, in favor of another IO to occupy the same space (presumably because its value to ongoing operations is greater) or in favor of maintaining the space free for occupation by future IOs (presumably because the *potential* value of the future IO to ongoing operations is greater). We chose to employ *fuzzy logic* to realize our

Value Depreciation Function because two properties of information depreciation and valuation match well to the principles underlying fuzzy logic:

- Whether an IO has sufficiently depreciated in value to move or not is not completely *true* nor completely *false*. Instead, it is more or less true or false depending on the other choices available, such as how bad the space is needed, what else there is to move, and what the information will be used for.
- The factors that go into determining an item of information's valuation lend themselves to relative interpretation. For example, whether an IO is old depends on the IO's age *relative* to that of other IOs. Whether moving an IO will free up much space depends on the IO's size *relative* to that of other IOs.

Fuzzy logic [7] originated in 1965, with the publication of "Fuzzy Sets" by L.A. Zadeh of the University of California, Berkeley, California [28]. Traditional sets typically are described using a binary membership function, *m*, where a set $S = \{x \mid m(x) = 1\}$, i.e., $m(x) = 1$ means that *x* is a member of the set and $m(x) = 0$ means that *x* is not a member of the set. An alternative way of expressing the traditional set *S* is as a pair, i.e., $S=(U, m)$, where *U* is the universe over which the set *S* can exist, and the function *m* determines the membership of any element, $s \in U$, in *S*. If $m(s)=1$, then $s \in S$. If $m(s)=0$, then $s \notin S$.

As an example, consider the set of all IOs in a Phoenix repository. The set *BFT* can be defined as the traditional set of all IOs that have the type, *BlueForceTrack*. That is, for a repository of IOs, *R*, *BFT = (R, f(i)=(type(i) == BlueForceTrack)).*

In contrast, consider defining all the sets of *large* IOs or *old* IOs. Although the *size* and *age* of each IO is quantitative, the judgment of whether something is *large* or *old* is a relative, fuzzy concept. These are not as well described by a traditional set, because of the traditional set's binary notion. For example, assume that the *large* set is defined as a traditional set over the universe of all IOs, with a membership function $f(i) = (size(i) > 100)$. An IO of size 101 would be in the set *large*, as would an IO of size 1000, and an IO of size 1,000,000. All of these IOs would have the same membership in the set *large*, despite the orders of magnitude differences in their sizes. Conversely, an IO of size 99 would not be in the set *large*, despite being much closer to the IO of size 101 than the other elements in the *large* set.

Fuzzy sets capture fuzzy, relative valued memberships better than the traditional sets. A fuzzy set is defined as a pair $F=(U, m)$ like the traditional set, but the function *m* in a fuzzy set is a function with a range in the interval [0,1], as shown in Figure 4. An element, $s \in U$, such that

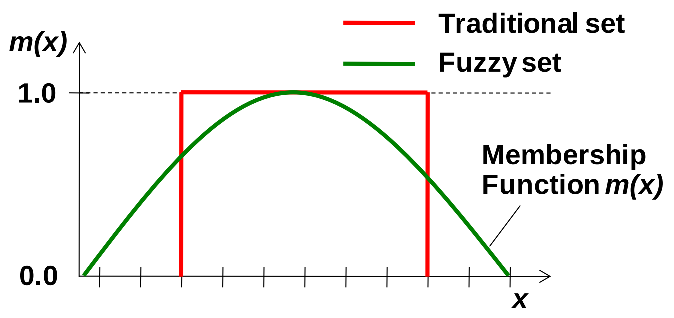


Figure 4. Traditional set membership vs. fuzzy set membership.

m(s)=0 still means that *s* is not a member of the set, i.e., $s \notin S$. Any non-zero value for *m(s)* indicates the *degree* to which *s* is a member of the set, with *m(s)=1* meaning that *s* is *fully* in the set.

In our size example above, the IOs of size 101, 1000, and 1,000,000 would each have a membership degree > 0 and ≤ 1, and the membership degree of the IO of size 1,000,000 would be larger than that of the IO of size 1000, which in turn would be larger than that of the IO of size 101.

In this way, a fuzzy set, *F=(U, m)*, provides a partial order over its members.

Fuzzy logic is a technique for making decisions based on combining the members of multiple fuzzy sets. It consists of the following three steps [26]:

- Acquiring a number of input values.
- Processing the inputs according to a set of fuzzy logic rules.
- Averaging and weighting the outputs of all the individual rules into a single output decision.

Fuzzy logic has been used in various applications. The subway system in Sendai, Japan uses a fuzzy logic controller to control the subway train's acceleration, slowing, and braking to ensure a smoother ride than position based controllers [12]. Fuzzy logic has also been used in air conditioning and heating system controllers [1], rice cookers [27], industrial automation [8], 3D Animation software [19], and elevator controls [10], [20].

The International Electrotechnical Commision (IEC) standardized the *Fuzzy Control Language, FCL*, in IEC 61131-7 in 1997 [11]. FCL enables the specification of fuzzy sets and "IF-THEN" rules. There are several software tools and packages available that implement FCL, including the following:

- jFuzzyLogic, http://jfuzzylogic.sourceforge.net/html/index.html.
- *fuzzy*TECH, by INFORM GmbH, http://www.fuzzytech.com/.
- The Free Fuzzy Logic Library (FFLL), http://ffll.sourceforge.net/.
- AwiFuzz, http://sourceforge.net/projects/awifuzz/.
- MATLAB's FuzzyLogic Toolbox, http://www.mathworks.com/products/fuzzylogic/.

3.7 Experimental methodology

As part of the VFILM project, we developed a set of metrics and conducted experiments to collect the metrics, described in detail in the *VFILM Experiment Plan* [15]. Our experimental methodology was the following:

- We developed metrics evaluating both the functionality and the performance of the VFILM prototype.
- We authored the experiments as JUnit tests so that they can be used for regression testing the software.
- Functional experiments contain assertions that fail on unexpected or incorrect results.
- Performance tests output logs and have scripts to extract relevant results into Comma Separated Value (CSV) files.
- In those experiments in which a baseline of comparison is needed, VFILM is compared against a baseline of the Phoenix Core Services with no information lifecycle management.

3.7.1 VFILM Metrics

Table 1 contains the functional metrics that we defined and gathered. These metrics answer the following questions about the correctness of the VFILM prototype system:

- Can the VFILM prototype system perform information valuation and movement in response to events (i.e., mission, system, and ILM events)?
- Can the VFILM prototype system maintain a threshold of desired space in the level 0 storage?
- Do archive and query operations have correct behavior when performed with Phoenix services using VFILM prototype functionality?

Table 1. Functional Metrics Defined for VFILM

No.	Description	Measured	Value, Units
F1	Responsiveness to events	Valuation and movement triggered by appropriate events	Yes/No
F2	Repository maintenance	Free space in level 0 store over the course of the experiment	% available graphed against time and threshold being maintained
F3	Correctness of archive operations	Publications with the archive bit set on baseline and VFILM code	The number of IOs from the published set that are in the archive at the end of the experiment in the baseline and experimental VFILM cases
F4	Correctness of query operations	Query operations on baseline and VFILM code	The number of IOs from the published set that are returned by the baseline and VFILM cases

Table 2 shows the performance metrics that we defined and gathered for VFILM. They address the following questions about the performance of VFILM:

- How does the valuation function, VDF, scale?
- How does the HSM movement of information scale?
- What overhead, if any, does VFILM introduce on the baseline Phoenix archive and query operations?
- What is the cost associated with the increased flexibility of the VFILM mission-driven information valuation and lifecycle management?

Table 2. Performance Metrics Defined for VFILM

No.	Description	Measured	Value, Units
P1	VDF scalability	Time to execute the VDF on a variety of repository configurations	Execution time of the VDF
P2	HSM scalability	Time to execute the HSM move operation on a variety of repository configurations	Execution time of the HSM move operation
P3	Performance of archive operation	Time from the start of publication to the last archive operation on baseline and VFILM code	Comparison of time to archive completion in the baseline and VFILM cases.
P4	Performance of query operation	Time to return query results on baseline and VFILM code	Comparison of the time to return all results in the baseline and VFILM cases.
P5	VDF execution time	Time to perform valuation using the current VDF and using a single-factor function.	Comparison of the time to execute the two valuation functions.

3.7.2 Experiment Definitions

To evaluate the metrics described in Section 3.7.1, we defined seven experiments with the following hypotheses:

1. *ILM Responsive to Events* – ILM valuation and HSM movement can be triggered by Phoenix and system events.
2. *Maintain Level 0 Store* – The ILM can maintain a specific amount of free space in Level 0 store.
3. *Correctness* – The results of archive and query operations on Phoenix with the VFILM prototype software will differ from the results of the baseline Phoenix operations only in the latency.
4. *Scalability (VDF)* – The time to evaluate objects increases linearly with the number of objects in the evaluation set.
5. *Scalability (HSM)* – The time for the HSM adapter to move IOs increases no worse than linearly with the number of objects moved and the total amount of bytes moved.
6. *Performance* – The time to execute archive and query operations should be largely unaffected by the presence of ILM functionality, except for the effects of retrieval from non-level 0 store.
7. *Cost of Valuation Flexibility* – The cost of using a Fuzzy Control Logic based valuation function is not prohibitively higher than the cost of some simple function, specifically IO age.

3.7.3 Experiment Infrastructure

All of the experiments were carried out on two specific computers.

The experiments for metrics P1, P2, P3, and P4 were run on a computer with a 1.7 GHz Quad-Core AMD Opteron(tm) Processor 2344 HE and 8GB memory.

The experiments for metrics F1, F2, F3, F4, and P5 were run on a computer with a 2.00 GHz 8-core Intel(R) Xeon(R) CPU E5405 and 4GB memory.

4.0 RESULTS AND DISCUSSION

The VFILM project produced the following significant research and development results:

A novel approach to information lifecycle management. Our approach to information valuation and movement based on fuzzy logic has several advantages over existing approaches, including the following:

- Multiple, non-traditional factors can be considered in valuing information urgency, including but not limited to mission factors, relation to other information, and information characteristics. The factors can differ between sets of information and the set of factors is extensible.
- It supports dynamic, event-driven information movement. The movement of information, and how much storage is needed, is not static. It can change based on the number and types of missions that are going on, and the nature of the information in the databases. Furthermore, the VFILM system supports mission, system, policy, and other relevant events through an easily extendable event-handler implementation.
- It avoids thrashing around fixed storage thresholds or drastic purges of information to free up storage. VFILM separates the information valuation and information movement functions, and treats information valuation as a partial order of the "criticality" of information. Each can be scheduled when needed or when resources are available, and can be executed as much as needed, e.g., to recover just enough space to continue.
- It provides a rich framework for specifying information valuation factors and policies for valuing and moving information. New Fuzzy Control Language rules, fuzzy sets, policies, groups, and thresholds are readily added or changed so that the VFILM system can be configured for many situations and uses.
- It can treat groups of information that are related collectively, so that they are valued and moved as a group, when appropriate.

A VFILM Architecture. We specified and documented an architecture for providing value-based information lifecycle management in the context of information management services. The architecture, which is described in Section 4.1, identifies the components that make up value-based information lifecycle functionality and their roles.

An ILM Service Design and Prototype Implementation. We designed an ILM service that works with AFRL's Phoenix IM services and implemented a software prototype, as described in Sections 4.2 and 4.3. The VFILM ILM service design and prototype provide the following significant features:

- Works with the existing Phoenix Repository and Query services (with minimal changes), and with multiple repositories spread over multiple storage levels.
- Separates information valuation and movement functions, and is extensible to specifying and modifying the factors that affect information valuation.
- Triggers information valuation and movement in response to events. The prototype works with Phoenix Events that specify changes in mission status, policy changes, and monitored system events. The ILM service is extensible to specify additional triggering events and event handlers.
- Supports groups of information that can be valued and moved collectively. Groups can overlap and can be based on any indexable information properties, such as type, source, mission, or location.

- Configurable and policy-driven, with the ability to specify information valuation rules, storage levels and thresholds, group membership, triggering events and event handlers, priorities, and precedence.

The prototype ILM service provides both a functional ILM functionality for Phoenix IM services and the Berkeley XML database, and a basis for more comprehensive and richer ILM functionality.

Demonstration and Evaluation of VFILM Capabilities. We provided a set of demonstrations of VFILM functionality and documentation to build and execute the demonstrations, described in Section 4.4. These include Graphical User Interfaces (GUIs) and demonstration clients. We also conducted a set of experiments collecting functional and performance metrics about the VFILM prototype, described in Section 4.5. The experiments are implemented as JUnit tests so that they can serve as a regression test suite for the software.

4.1 VFILM Architecture

ILM for Air Force enterprise and tactical environments requires a mission-driven, flexible, and intelligent *ILM service* for deciding when information should move between levels, what information should move, and for managing the following aspects of the information lifecycle:

- *The value of information* and the factors that can contribute to an increase or decrease in information value. Most ILM and HSM approaches available rely on recentness or frequency of information access to determine where information should reside [2]. This is insufficient in military situations because it ignores the inherent *value* of information and its *criticality* to ongoing operations. The access patterns of an IO contribute to its value, but are not the only factors. Mission-related information can be of higher or lower value based on its *urgency* to a mission and *how rapidly* it must be provided if accessed (necessitating its availability in local store).
- *The grouping of information*, so that related information can be moved and/or retrieved together when appropriate. Traditional ILM and HSM focus on files and documents, an 80% solution appropriate for most business needs. VFILM supports a richer organization of information, including instances of Phoenix Information classes and grouping related and derivative information, e.g., based on association with a mission, sortie, or platform; common attributes (e.g., age); and other aspects.
- *The triggers of information movement*, which for existing HSM solutions are generally limited to when space becomes needed or specific scheduled times. This is not sufficient for USAF needs because when space is reaching capacity might be exactly when additional space is most needed by the mission and might be when the time and resources needed to move information to free up space are least available. We utilize a rich set of triggers to invoke the ILM to decide whether information should be moved, how much, and when, including the need for space, events such as mission epochs, and proactively when the ILM and HSM functions would not impact mission operations.

The view that ILM and HSM are *management services,* not simply a process with an HSM mechanism, matches the service-oriented and active management organization of AFRL IM thrusts. This facilitates a mostly automated technical solution rather than an expensive business process provided by a specific vendor, and supports various HSM solutions.

The VFILM architecture, shown in Figure 5, consists of an ILM service that assesses information value based on urgency to current mission needs, invokes information movement when

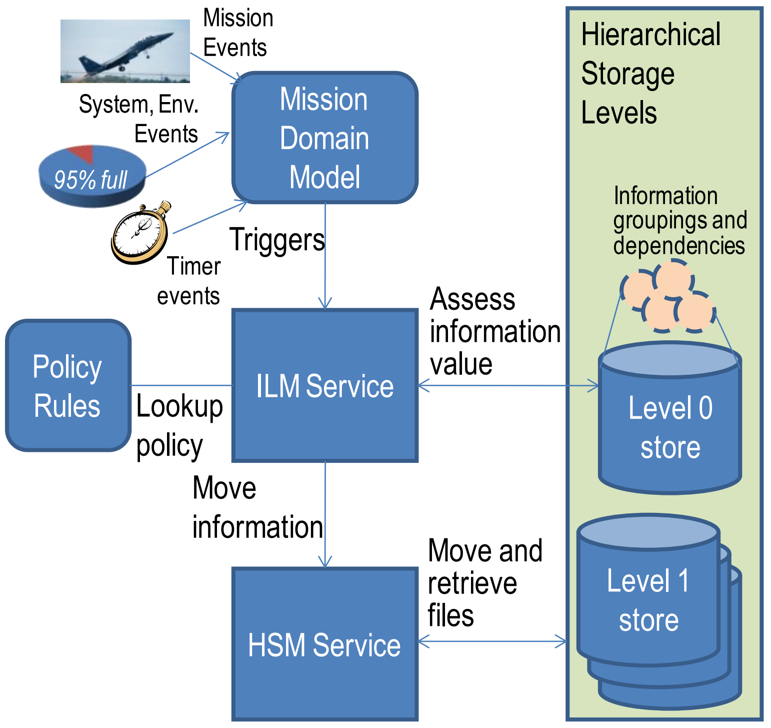


Figure 5. The VFILM architecture.

needed, and controls an HSM service that moves information between hierarchical storage levels. It consists of the following core architectural elements:

- The *ILM service*, which decides when and what information should be moved.
- A *value depreciation function* that determines how information value changes in urgency, criticality, or importance.
- *Events* that serve as *triggers* for information valuation and movement. These include *mission events* such as the start and end of mission operations, *system events* such as memory exhaustion, and *timer events* such as a garbage collection timeout.
- *Mission domain models* that map mission events to ILM operations.
- *Policy* governing information movement and retention, thresholds on storage limits, and how groups are treated.
- *Grouping* that captures information relationships and mission dependencies and enables information to be moved or retained collectively.
- The *hierarchical storage levels* of storage media in which information can reside, including *level 0*, the local disk where the IO repository resides, and backing store (levels 1+).
- An *HSM service* that performs information movement and retrieval.

4.2 VFILM Prototype Design

In addition to defining the VFILM architecture, we also designed and implemented a prototype of the VFILM ILM service and VDF algorithm. The ILM design and prototype implementation that we developed includes enough functionality for other components of the architecture to make the ILM functional and useful in the AFRL Phoenix IM context, but full instantiations of a

Mission Domain Modeling component and an HSM Service were determined to be out of scope for this project. We investigated a number of HSM services, as described in Section 3.5, and could find no existing HSMs that were mature, accessible, supported, platform-independent, and cost-effective enough to utilize, so we prototyped a representative set of HSM capabilities for moving information objects between hierarchical storage levels.

Table 3 lists the architectural elements from Section 4.1, summarizes their design within the ILM service, and the section in which they are described.

Table 3. Design of prototyped VFILM components.

VFILM Architecture Element	Prototype Design	Section Described
ILM Service	A new service optionally deployed with Phoenix IM services.	4.2.1
Events	The Event Manager component of the ILM service and pluggable event handlers, with sets of defined ILM events, mission events, and a file system monitor for system events.	4.2.2
Mission Domain Model	The Default Mission Domain Model Event Handler.	4.2.3
VDF	Fuzzy logic based algorithm within the ILM service.	4.2.5
Grouping	The Group Manager component of the ILM service.	4.2.6
Policy	Event handlers, fuzzy logic rules, and configuration files.	4.2.7
HSM Service	ILM-HSM adapter and simulated HSM functionality.	4.2.8

4.2.1 Design of the ILM Service

The design of the prototype ILM service that we developed is shown in Figure 6 and consists of the following components:

- The *ILM Event Manager* manages event handlers that create ILM events in response to Phoenix events and other inputs.
- The *ILM Controller* drives the behavior of the ILM in response to ILM Events.
- The *Value Depreciation Function* evaluates information objects using a specified policy.
- The *Group Manager* maintains the definitions of groups of information.
- The *ILM-HSM Adapter* abstracts away the specifics of the HSM and Phoenix Repositories being used.

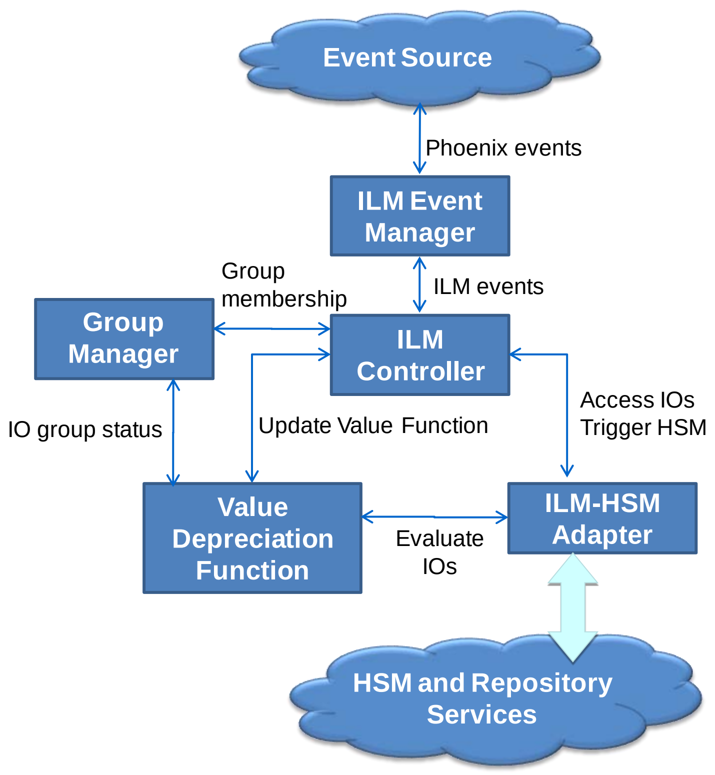


Figure 6. Design of the ILM Service.

4.2.2 Design of the ILM Event Manager

As shown in Figure 7, the ILM Event Manager maintains a set of Event Handlers, each of which receives incoming higher level events and maps them to ILM events understood by the ILM Controller. The Event Handlers are pluggable. We prototyped a set, but additional ones can be provided at configuration-time or at runtime (see the Policy Event Handler below).

Generated events and discrete epochs, such as mission events and policy events, are represented as Phoenix events and delivered using Phoenix Event Channels. The consumer of the Phoenix Events is the Event Manager that selects the appropriate event handler to use for each event, based on the event type. Continuous conditions, such as the amount of free storage, can be monitored directly by event handlers.

Each event handler maps the incoming or monitored higher-level events to a set of ILM events, and the set of ILM events is passed to the ILM controller for execution.

All of the ILM actions are driven by a set of ILM events that serve as the “language” of the ILM and *trigger* the ILM to conduct information valuation, information movement, group updates, and/or policy modification. The following is the set of prototyped ILM events:

- *NeedSpace* – Indicates that a particular amount of space should be made available through the movement of information.
- *Cleanup* – Check the relative valuation of information across the hierarchical levels and *rebalance* the location of information, so that the most critical information (lowest depreciation valuation) is in level 0 store and less critical information (highest depreciation valuation) is in higher storage levels.

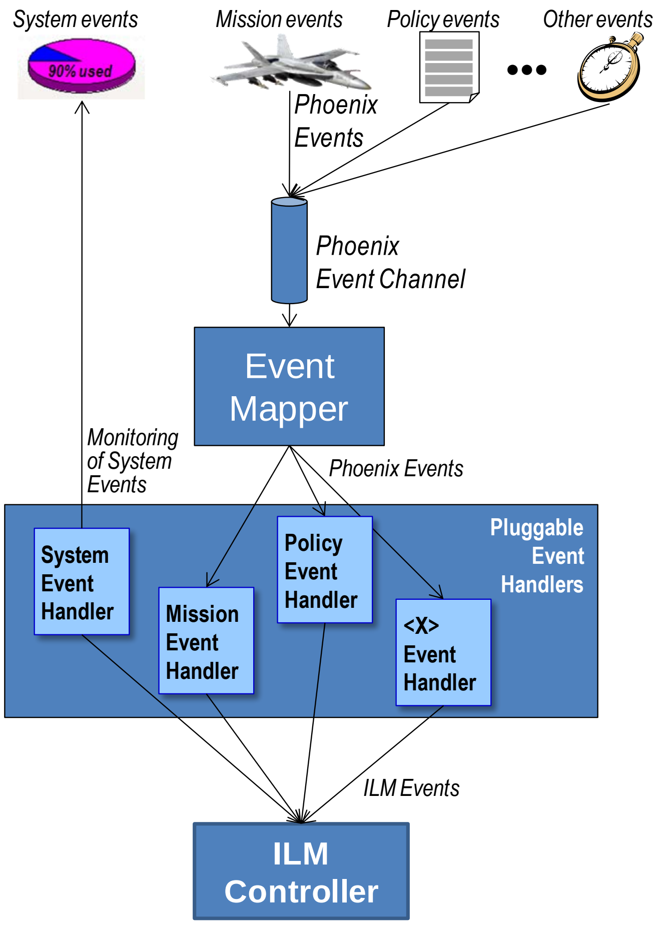


Figure 7. Design of the ILM Event Manager.

- *UpdateThreshold* – Change the threshold of space that the ILM should maintain available in level 0 storage.
- *Valuation* – Execute the VDF valuation function on a set of information (provided as a parameter) to determine the information valuation.
- *GroupUpdate* – Create a new group or change the attributes of a group of information objects.
- *RuleChange* – Add or change a fuzzy logic rule determining the valuation of information objects.
- *MoveIOs* – Move a set of information objects from one repository to another (usually in different storage levels).

The VFILM prototype provides the following five Event Handlers:

- *Default Mission Domain* – Reacts to incoming Mission Events, described in more detail in Section 4.2.3.
- *File System Monitor* – Monitors the level of free space and triggers a *Need Space* event when the available space drops below a specified threshold, as shown in Figure 7.

- *ISQM Listener* – Implements the ISQM Listener Interface used with the Quality of Service (QoS) Enabled Dissemination (QED) prototype [17] to receive policies associated with groups of information (e.g., missions or information types).
- *Policy Handler* – Provides ILM administration, such as setting the free space threshold or inserting new Event Handlers.
- *Location Manager* – An Event Handler created for the VFILM demonstration. It subscribes to track data published by moving clients and triggers group and valuation events as the location of the client changes. Described in more detail in Section 4.4.

4.2.3 Design of the Mission Domain Model

The prototype Mission Domain Model for VFILM is provided by the Mission Domain Model Event Handler, the set of mission events that we defined for the VFILM prototype, and the mapping to ILM events. We defined the following Mission Event Types for the VFILM prototype:

- *MissionPrep* – Indicating that a planned mission will start sometime in the future and the ILM should prepare for it.
- *MissionBegin* – Indicating the start of a mission.
- *MissionEnd* – Indicating the end of a mission.

The Mission Domain Model Event Handler maps these three Mission Event Types to the ILM Events indicated in Table 4.

Table 4. Mission Events and mapping to ILM Events representing the prototype VFILM Mission Domain Model.

Mission Event Type	ILM Events	Resulting ILM operations
MissionPrep	Cleanup	Runs the valuation function on multiple storage levels and sorts the results so that the IOs are balanced across the storage levels according to their valuation and the storage thresholds.
MissionBegin	GroupUpdate	Creates a group representing the mission.
	Valuation	Triggers valuation of all IOs matching the mission predicate.
	NeedSpace	Moves IOs to free up enough available space for the mission.
MissionEnd	GroupUpdate	Removes the group associated with the mission.
	Valuation	Triggers valuation of all IOs associated with the mission, i.e., matching the mission predicate.

4.2.4 Design of the ILM Controller

The ILM Controller receives ILM Events from the Event Handlers and invokes valuation, movement, or update functions as indicated in Table 5.

Table 5. ILM Events implemented by the ILM Controller and their evaluation.

ILM Event	Parameter	Description of algorithm
NeedSpace	Amount of space *X*	Move IOs until there is *X* amount of space in level 0
Cleanup	Amount of space *X*	Sort IOs into storage levels by their valuation, leaving *threshold* + *X* bytes of free storage in level 0
UpdateThreshold	Threshold *X*	Change the *threshold* value the ILM maintains in level 0 to the value *X*
Valuation	Information set *X*	Invoke VDF on the IOs in *X*
GroupUpdate	Group context *X*	Update the context associated with a group
RuleChange	Evaluation rule *X*	Add *X* to the set of evaluation rules
MoveIOs	*src*, *dest*, *# IOs X*	Move *X* IOs from *src* repository to *dest* repository

4.2.5 Design of the Value Depreciation Function

The Value Depreciation Function determines whether an information object's usefulness (or value) has *depreciated* enough to move from level 0 store to backing store or, conversely, whether it has *appreciated* and therefore needs to be retrieved from backing store and re-inserted into level 0 store.

Whether information should be moved out of level 0 store comes down to a difficult to quantify predictive measure, i.e., whether the information will be needed soon (or ever). Furthermore, it is a relative assessment, i.e., whether the space in level 0 store occupied by an IO *X* is best used for *X* or a different IO, or left available for future information. Furthermore, there can be multiple factors that go into deciding the relative worth of information objects, including the missions or operations that they are being used in (indicating how relevant they are to ongoing operations), the age of the IOs (indicating how fresh the information is), and the size of the IOs (indicating how much space they are using). Each of these factors has relative interpretation. That is, whether an item of information is relevant enough to keep, or large or old enough to move is relative to other items of information and to the anticipated use of the space if the information object is moved. Any discrete or static threshold for any of these factors will lead to inflexibility, i.e., it is likely to only be suitable in specific situations and not sufficient in others, and potential thrashing.

Therefore, VFILM takes the approach of building a *partial order* of information valuation so that at any time when information needs to be moved to make room in level 0, the information that is most *depreciated* in value relative to the others will be moved.

We use a *fuzzy logic* rule based approach to produce the partial order from relative valued inputs. As shown in Figure 8, information factors such as mission relevance, age, and size are expressed as fuzzy input sets. Figure 8 shows three fuzzy inputs, clockwise from lower left corresponding to *age*, *mission relevance*, and *size*. Each of these fuzzy inputs consists of multiple sets, e.g., *age* consists of a *Young* and an *Old* fuzzy set. The *x axis* represents the measured value of the input, e.g., the age of an IO calculated from the current time and a creation timestamp, and the *y axis* represents the degree of membership in a particular fuzzy set.

Figure 8 also shows that a set of fuzzy logic rules specifies how to combine these inputs into a degree of membership in an output set, *Move*. The degree of membership in the *Move* set defines the partial order of information valuation. When space is needed, the IOs with the highest degree of membership in *Move* are the ones chosen to be moved.

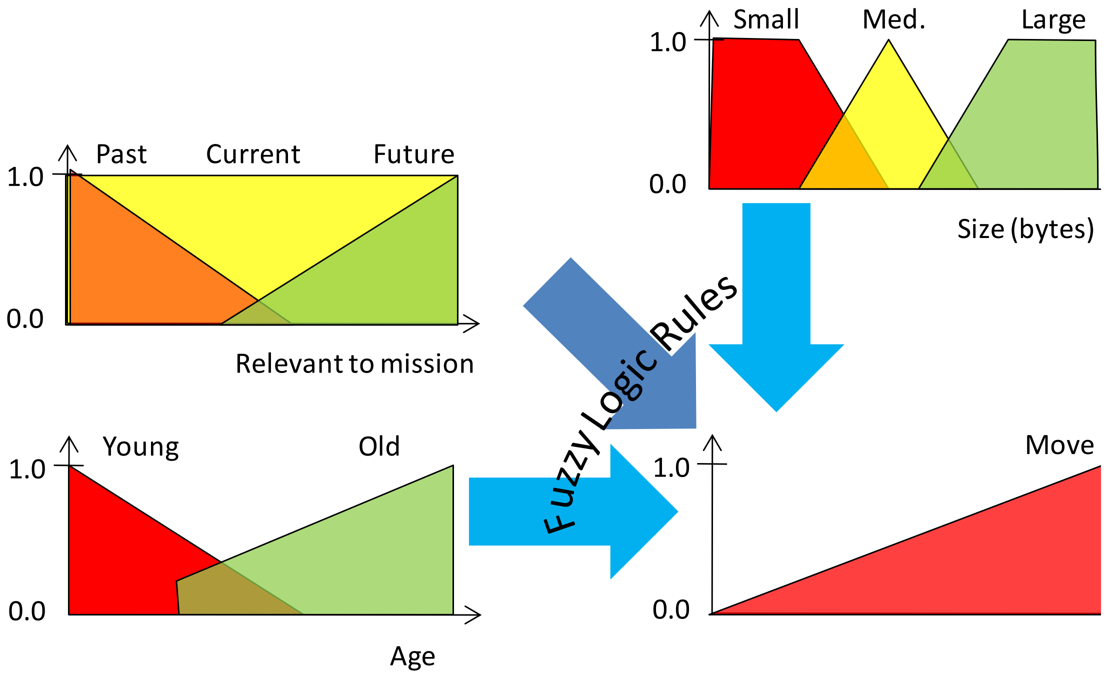


Figure 8. The combination of fuzzy input sets into relative membership in a *Move* set using fuzzy logic rules.

This design offers a tremendous amount of flexibility and extensibility in the VDF. New factors can be added to the valuation by introducing a new fuzzy input set. The rules for combining the fuzzy inputs into the output measure can be extended. Finally the various input factors can be weighted so that some factors contribute more to the output set than others.

The design of the VDF ILM component consists of the following pieces:

- Fuzzy sets representing the inputs and output of the VDF function.
- Fuzzy logic rules that combine the inputs into a degree of membership in the output set.
- Functions that access the values for the fuzzy inputs, which can be stored in information metadata, Phoenix Context objects, system condition monitors, operating system attributes, etc.

The implementation of the VDF component and these pieces are described in Section 4.3.6.

4.2.6 Design of the Group Manager

In many cases, IOs are not independent entities and there are significant advantages to having the ILM exploit the interdependencies. One realization of information interdependencies is association with a common group. As shown in Figure 9, information in a system can be associated with many overlapping

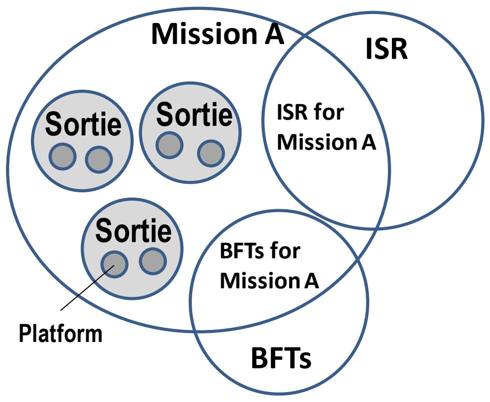


Figure 9. Information organized into many groupings, some of which have associated lifecycles.

and co-existing groups, based on shared types (e.g., *blue force tracks, BFTs*), source (e.g., a specific platform), role (e.g., intelligence, surveillance, and reconnaissance, *ISR*, for mission A), epoch (e.g., a sortie), location within a particular region, and so forth. VFILM supports the association of IOs that are related and that should be treated collectively into groups. Events can affect a group of IOs and IOs can be collectively valued and moved.

Groups are defined using predicates over observable attributes, such as IO type, metadata, or attributes on contexts. We defined a new *Group Context*, shown in Figure 10, to hold the following information about a group:

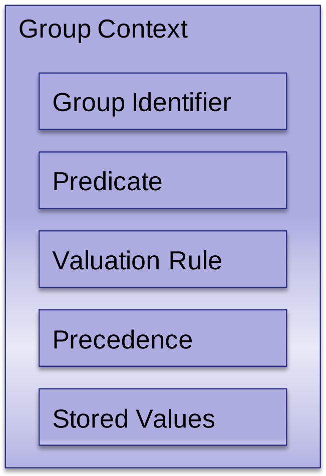


Figure 10. The Group Context contains the information needed to represent a group of IOs.

- *Identifier* – A name to identify the group.
- *Predicate* – A predicate defining the IOs in the group. The predicate is defined over fields in metadata, contexts, or other information derived from an IO.
- *Valuation rules* – The set of rules that is used to evaluate the IOs in the group.
- *Precedence* – Used to determine which group definition is used during IO valuation when an IO is part of multiple groups.
- *Stored values* – Input values associated with a group and used during IO valuation.

The *Group Manager* maintains the collection of *Group Contexts*. Missions are represented as just another type of group. When a Mission Start event occurs, a Group Context is created for IOs associated with the mission.

4.2.7 Design of VFILM Policy

There are a number of VFILM elements that collectively make up the VFILM policy governing information valuation, movement, and attributes affecting the configuration of the system and the factors that go into information valuation and movement.

An explicit element of policy that we added is support for QED-like policy. QED is another project that BBN led that created policy-driven QoS management for Phoenix and includes a policy language for specifying mission- and client-driven QoS policies [17]. The VFILM prototype design includes a QED-like policy handler, which receives policies of the form

Policy : $f(o,m) \rightarrow v, i, P;$

Where o is the operation to which the policy pertains, m is observable attributes of information, v is the precedence, i is an importance, and P represents a set of preferences.

Whereas QED supports multiple IM operations (o), VFILM is only concerned with the *query* operation. The condition element, m, can include attributes of information such as its type, or other attributes that can be indexed. The precedence, v, maps to the *Precedence* stored in the Group Context and used to deconflict multiple groups when performing valuation of IOs. The importance, i, is a special group attribute that can be used as an input value to the fuzzy logic valuation function. Importance is used, for example, to identify that one Mission group is more important than another Mission group. The preferences, P, specify a set of *name-value* pairs. In the VFILM prototype *P* is not being utilized, however, the *P* name-value pairs could easily be treated as attributes in the Group Context and their values, and used as inputs to the valuation function.

Similar to reusing the QED policy in the VFILM context, we decided to use other existing features to implement policy, rather than introducing a special purpose VFILM policy language. This enabled us to maintain the VFILM services as transparent and non-intrusive to the Phoenix services as possible, without introducing another interface (a policy language) that would need to be learned to use the VFILM prototype. Therefore, VFILM policy is encapsulated in all of the following:

- VDF Evaluation Rules that specify how information is valued.
- Group details stored in Group Contexts and specified in QED-like policy.
- Storage thresholds provided via configuration files and policy events.
- ILM Event triggers, including the Mission Domain Model, File System Monitor, and other Event Handlers.

4.2.8 Design of the ILM-HSM Adapter

The ILM-HSM Adapter is a control interface from the ILM to HSM functionality that is intended to support a variety of HSM implementation options. As such, it provides a consistent interface for the ILM to specify IOs and files that should be moved independently of the specific HSM or repository that is used. In a situation where a full HSM solution is not appropriate, the ILM-HSM Adapter can be responsible for the movement of information. If the repository stores IOs as files on disk this could involve moving the file and updating the repository’s reference or leaving a symbolic link to the file’s new location. With other repository implementations where IOs are stored in relational databases, such as with the PostGIS Repository, this could involve removing the IO from one table and inserting it into another.

In situations where an HSM is used, the ILM-HSM adapter would serve as the interface to the HSM. This will depend on the specifics of the HSM utilized but could involve assigning priority values to managed files based upon information value or modifying a management policy.

The ILM-HSM Adapter also provides the ILM with access to IOs stored in the repository.

Since our investigation of off-the-shelf HSMs did not turn up anything suitable for our VFILM prototype efforts (as described in Section 3.5), we simulated HSM functionality within the ILM-HSM Adapter. We designed it to take advantage of, and be consistent with, the Phoenix use of the Berkeley DB. The ILM-HSM can handle multiple repositories, can move just IOs retaining metadata in the level 0 store, or move metadata and IOs to level 1 store.

The ILM-HSM Adapter maintains the following two extra databases on each level of storage to facilitate information movement:

- A *Value Store* – Contains the IO context ID, IO value (i.e., the result of the most recent valuation execution), and the storage level of the IO.
- An *ILM Index* – Maintains an index of the IOs in the level of storage indexed by the fields used to define group membership allowing rapid lookup of IOs associated with a group.

Each type of repository has a corresponding ILM-HSM Adapter type, and each repository instance has an ILM-HSM Adapter instance of the corresponding type, as shown in Figure 11. The ILM-HSM Adapter provides multiple options for moving IOs. The first option moves the IOs only, retaining the metadata in level 0 store, so that the Phoenix Query Service works the same as before. That is, it matches the metadata in the Metadata DB and follows the pointer to retrieve the matched IO, where the IO might be in level 0 or in level 1 store.

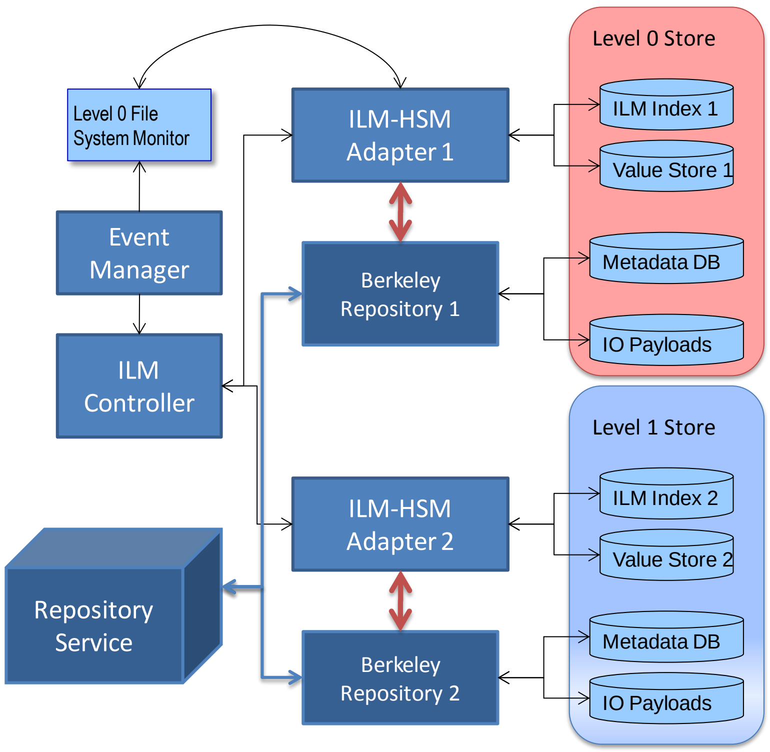


Figure 11. Design of the ILM-HSM Adapter.

The ILM-HSM Adapter design for moving IO files only (no metadata) from level 0 to level 1 is shown in Figure 12 and works as follows:

- When an IO should be moved, the ILM-HSM adapter physically moves the IO's file from the Level 0 filesystem to the Level 1 filesystem, placing the IO under HSM control.
- The ILM-HSM adapter then updates the file reference for the IO in the MDDB to reflect the new location.

Because retaining all of the metadata in level 0 store can still lead to the level 0 store filling up, the ILM-HSM Adapter can also move metadata and IOs together. As shown in Figure 13, the movement of IOs only and the movement of metadata and IOs together can coexist.

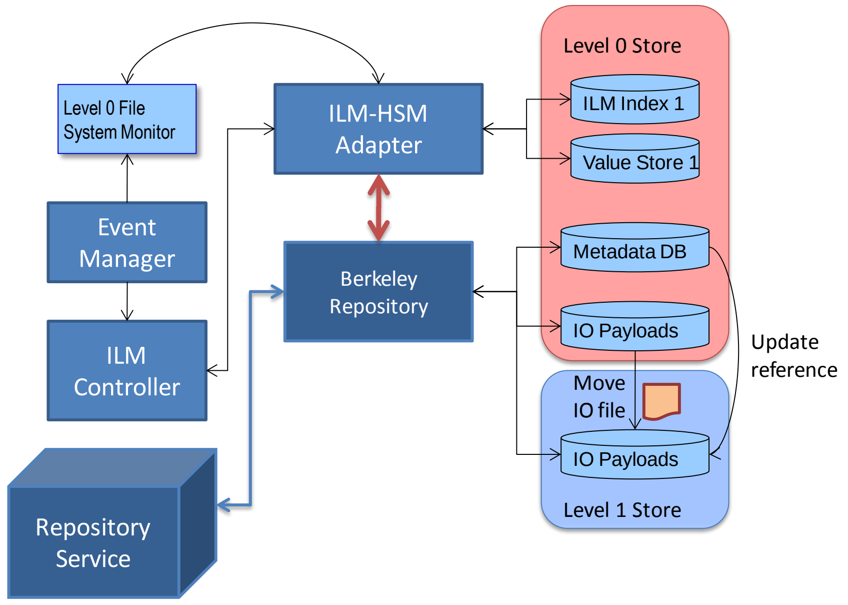


Figure 12. The ILM-HSM can move IOs from level 0 to level 1, retaining the metadata in level 0 with updated references to IOs in level 1.

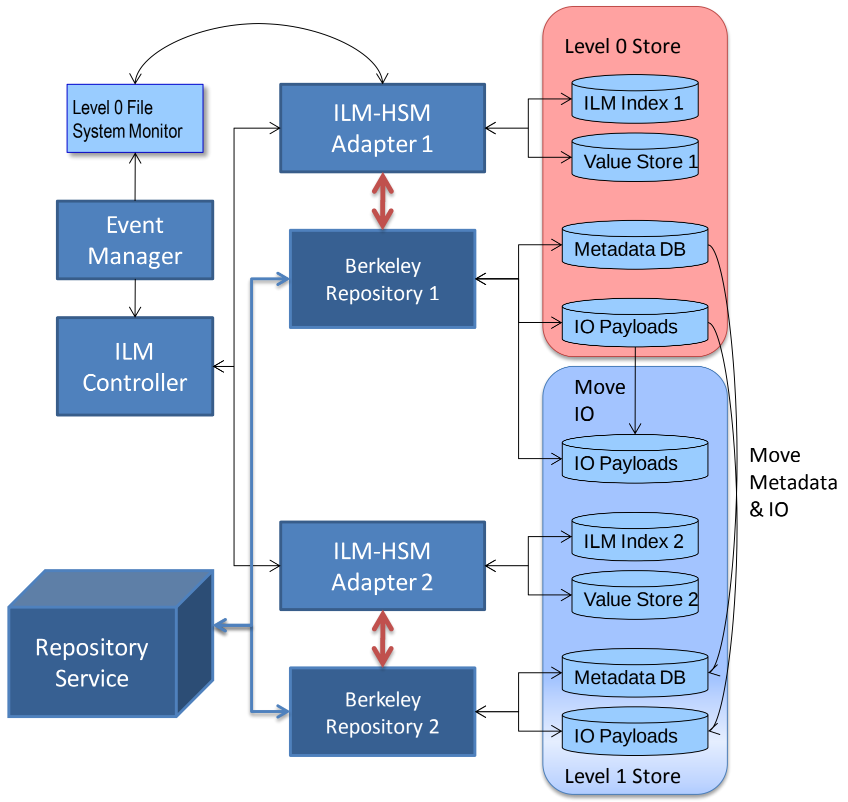


Figure 13. The ILM-HSM can move metadata and IOs together from level 0 to level 1.

4.3 VFILM Prototype Implementation

The VFILM prototype implements an ILM Service, consisting of an Event Manager, Controller, Group Manager, ILM-HSM Adapter, and Value Depreciation Function, as shown in Figure 6 and described in Table 3. The following sections describe the implementation of each component.

In addition, we made minor changes to the Repository Service and Berkeley Repository to support the prototype implementation. These are described in Sections 4.3.2 and 4.3.3.

4.3.1 Prototype Implementation of the ILM Service

The ILM Service (`mil.af.rl.phoenix.ilm.service.ILMService`) extends the Phoenix Base Channel Service (`mil.af.rl.phoenix.channel.service.BaseChannelService`) and implements the ILM Service Interface (`mil.af.rl.phoenix.ilm.service.ILMServiceInterface`). Its only constructor argument is an ILM Service Context, which extends Service Context and implements the ILM Service Context Interface.

The ILM Service Context includes the following:

- Internal ILM Components
    - The Value Depreciation Function
    - The ILM Controller
    - The ILM Event Manager
    - The Group Manager
    - Optional GUIs
- External Phoenix Services
    - A reference or connector to the Event Notification Service (or appropriate connector)
    - Channel context for incoming events from the Event Notification Service
    - A reference or connector to the ILM Compatible Repository Service

All ILM-HSM Adapters are stored in a mapping (`Map<String, AdapterInterface> adapterMap`) of repository names to the appropriate ILM-HSM Adapter. Each adapter manages its own ILM Index and Value Store.

When the ILM Service connects to an ILM Compatible Repository Service, it retrieves a mapping (`Map<String, BaseContextInterface> repositoryContextMap`) from repository UIDs (as defined by the Repository Service) to RepositoryContexts. The appropriate adapter is then created for each repository context (if possible) via a call to

```
public void addRepository(ILMRepositoryContext repoContext, String repoUID)
```

and added to the adapter map. Currently only Berkeley Repositories are supported via the Berkeley Repository Adapter `mil.af.rl.phoenix.ilm.adapter.BerkeleyRepoAdapter`.

The ILM Service also registers an incoming Event channel with the Event Notification Service. ILM Event Handlers can register subscriptions for specific event types via a call to the ILM Service's

```
void registerEventType(String eventType)
```

Incoming events are received by an extension of a Timer Based Buffer (`mil.af.rl.phoenix.ilm.service.ILMTimerBasedBuffer`) and are passed to the Event Manager via a method call to

```
void process(EventInterface phoenixEvent)
```

4.3.2 Modifications to the Repository Service

In order to manage IOs located within repositories, the ILM Service needs additional access to the Repository Service beyond that provided by the baseline Phoenix implementation. For this reason, we created the ILM Compatible Repository Service (`mil.af.rl.phoenix.repository.service.ILMCompatibleRepositoryService`). This service extends the baseline Phoenix Repository Service and implements the ILM Compatible Repository Service Interface (`mil.af.rl.phoenix.repository.service.ILMCompatible-RepositoryServiceInterface`).

4.3.2.1 ILM Compatible Repository Service

We added the following six methods as an extension to the Repository Service.

```
public Map<String, BaseContextInterface> getRepositoryContextMap()
```

Returns a map containing the UID and context of each repository instance the Repository Service is managing. This is used for configuring an ILM-HSM adapter for each repository.

```
public long moveIO(String repositoryUID, String informationContextId, int
targetStore)
```

Moves the IO specified by `informationContextId` in the repository specified by `repositoryUID` to the target storage level `targetStore`. Returns the number of bytes moved or -1 in case of error.

```
public InformationInterface[] getIO(String repositoryUID, List<String>
ioContextIdList)
```

Retrieves the IOs specified by context ID from a specific repository.

```
public void insertInformation(String repositoryUID, InformationInterface[]
informationArray)
```

Inserts the provided IOs into the specified repository.

```
public void deleteInformation(String repositoryUID, List<String>
informationContextIdList)
```

Removes the specified IOs from the specified repository.

```
public void setILMNewInformationChannel(String repositoryUID,
ChannelContextInterface channelContext)
```

Configures an Information Channel for IOs inserted into the repository back to the ILM-HSM Adapter.

4.3.2.2 Modifications to the Berkeley Repository

Additionally, we made several changes to the Berkeley Repository (`mil.af.rl.phoenix.repository.impl.BerkeleyRepository`). Specifically, we changed `informationTable` from `HashMap<String, FileRepository>` where the key was an Information Type and the entry was a File Repository, to `HashMap<String, List<FileRepository>>` where the key is still a information type, but now the entry is a list of File Repositories, with one located on each storage level. Similarly, we changed `File informationDir` to `List<File> informationDirList` with an information directory for each storage level.

Additionally, we added the necessary methods so that the BerkeleyRepository implemented the ILM Repository Interface (`mil.af.rl.phoenix.ilm.adapter.ILMRepositoryInterface`).

The ILM Repository Interface defines the methods necessary for a repository implementation to be compatible with our implementation of the ILM-HSM Adapter. This interface consists of the following methods:

```
public long moveIO(String contextId, int targetStore)
```

Moves the specified IO to the specified storage level, returns the number of bytes moved.

```
public List<File> getInformationDirList()
```

Returns the list of directories storing IOs.

```
public InformationInterface[] getIO(List<String> contextIdList)
```

Returns the IOs with the given Context IDs.

```
setILMNewInformationChannel
```

Configures an Information Channel between the repository and the ILM-HSM Adapter, this is used for new IO's added to the repository so that the ILM can run the valuation function on them and manage them.

```
public boolean isArchival();
```

Indicates if this repository instance is used primarily for archival purposes. See Section 4.3.2.5 for more details.

```
public void deleteIos(List<String> contextIdList);
```

Removes the specified IOs from this repository.

```
public int[] getMetadataStorageLevels();
```

Returns an integer array consisting of the levels on which this repository stores metadata (the current implementation of the Berkeley Repository only stores metadata on one level and the returned array will be of size 1).

```
public void setMetadataStorageLevels(int level[]);
```

Sets the levels on which this repository stores metadata, used for configuration purposes.

```
public int[] getPayloadStorageLevels();
```

Returns an array containing the levels on which this repository stores payload information.

```
public void setPayloadStorageLevels(int[] range);
```

Sets the levels on which this repository stores payload information, used for configuration purposes.

4.3.2.3 Modifications to the File Repository

The BerkeleyRepository utilizes the FileRepository (`mil.af.rl.phoenix.repository.impl.FileRepository`) for storing IO payloads on disk. We added the following method to the File Repository implementation allowing IOs to easily be moved from one storage level to another.

```
public File moveInto(File oldLoc)
```

This method moves a File from its previous location into a directory managed by this File Repository, updating the directory and file counts as it does. **Note**: The current implementation of this method utilizes the Unix move command "mv" for the actual movement of the file. This call is likely not recognized by some operating systems, specifically MS Windows based platforms, and would need to be updated for cross platform compatibility.

4.3.2.4 ILM Query Context

Storing information on multiple storage levels opens up more options for queries. For example, it may be desirable to specify that a query should only be run over metadata on certain storage levels, and that results should be returned from other locations. To accommodate this we created an ILM Query Context (`mil.af.rl.phoenix.query.ILMInformationQueryContext`) which extends the Phoenix Query Context (`mil.af.rl.phoenix.query.InformationQueryContext`). The ILM Query Context contains a range of levels over which to query and a range of levels from which to return results.

It is important to note that the ILM Query Context will act as a normal Query Context class if the query is sent to a Repository that is not ILM Compatible. If a standard Query Context is sent to an ILM Compatible Repository, it will execute and return results spanning all levels. This allows us to achieve the desired functionality and remain compatible with baseline Phoenix implementations.

Within the Berkeley Repository, we made changes to compare what storage level its metadata is stored on before running the query, and then to check the where the payload of results are stored before retrieving them.

4.3.2.5 Repository Instances for Archival Purposes

The VFILM Prototype is capable of utilizing multiple repositories located on multiple storage levels for moving and managing IOs. However, the default behavior for the Phoenix Repository Service is to insert new IOs into all repositories, which could potentially result in undesired data replication. To prevent this with HSM repositories, we added an "`isArchival`" flag to the ILM Repository Context. When a new IO is being added to the Repository Service, the Repository Service will check if each repository instance uses an ILM Repository Context. If it does and the ILM Repository Context has the "`isArchival`" flag set to `true`, the Repository Service will not insert the IO into that repository. Instead IOs will be added to that repository via the ILM Service.

4.3.3 Prototype Implementation of the ILM Event Manager

The ILM Event Manager (`mil.af.rl.phoenix.ilm.eventmanager.EventManager`) implements the Event Manager Interface (`mil.af.rl.phoenix.ilm.eventmanager.EventManagerInterface`) and maintains a collection of ILM Event Handlers. The ILM Event Handlers trigger ILM actions in response to external events such as a system event indicating a storage quota is exceeded, a Phoenix event indicating a new mission is beginning, or a policy change.

One source of external events are incoming Phoenix Events. When a Phoenix event is received, the event manager passes it to the appropriate event handler based on a mapping (`Map<String, List<IlmEventHandlerInterface>> handlerMap`) of event type to lists of handlers that can process that type of event. The event handlers then use the content of the Phoenix Event to construct a list of ILM Events that are passed to the ILM Controller. For example, a Mission Event could be passed to the Default Mission Domain Model, which in the case of a mission start would trigger *Group Update*, *Valuation*, and *Need Space* events.

Event Handlers are not limited to being triggered by Phoenix Events. For example, once initialized, the *FSMonitor* spawns a thread that monitors a specific storage location. If the amount of free space at that location drops below a certain threshold, the *FSMonitor* sends a *NeedSpace* event to the ILM Controller.

4.3.4 Prototype ILM Event Handler Interface

The ILM Event Handler Interface (`mil.af.rl.phoenix.ilm.eventmanager.IlmEvent-HandlerInterface`) provides the general framework for all ILM Event Handlers. The ILM Event Handler Interface contains the following three methods:

```
public String getEventType()
```

Returns the type of Phoenix events this handler can process. This is used by the Event Manager while building the handler map.

```
public void initialize()
```

This method is called when an Event Handler is being started. Some may spawn a thread or initiate a connection. Others, such as the Default Mission Domain Model, register a new predicate with the Event Notification Service so that the ILM Service receives Phoenix events of certain types.

```
public void processEvent(EventInterface event)
```

The method called by the event manager when an incoming Phoenix event is received. This method can be empty for some event handlers. For example, the File System Monitor does not react to incoming Phoenix Events, and only reacts to changes in system state.

The VFILM Prototype currently has six implemented ILM Event Handlers which provide for a range of different ILM behaviors, these are described in the subsections below. Each section indicates the Event Handler's implementation class, interface class, and the class that it extends. Each section also indicates the implementation of the three methods described above, i.e., the `eventType`, the behavior of the `initialize` method, and the behavior of the `processEvent` method.

4.3.4.1 File System Monitor

Implementation:	`mil.af.rl.phoenix.ilm.eventmanager.FSMonitor`
Interface:	`mil.af.rl.phoenix.ilm.eventmanager.IlmEventHandlerInterface` `Java.lang.Runnable`
Extends:	`mil.af.rl.phoenix.ilm.eventmanager.AbstractEventHandler`

The File System Monitor implements the basic HSM functionality of monitoring available disk space. It periodically checks the amount of free space available to a specific ILM-HSM Adapter and compares the amount the storage thresholds set for that Adapter. If the amount of free space drops below the specified threshold it triggers a *NeedSpace* event for that repository.

```
eventType: null

initialize: Spawns the thread which continuously monitors an adapter's storage location.

processEvent: The File System Monitor does not respond to Phoenix Events.
```

Note: The implementation of the File System Monitor requires it to have access to the directories in which a repository stores information.

4.3.4.2 Default Mission Domain Model

Interface:	`mil.af.rl.phoenix.ilm.eventmanager.IlmEventHandlerInterface`
Implementation:	`mil.af.rl.phoenix.ilm.eventmanage.DefaultMissionDomainModel`
Extends:	`mil.af.rl.phoenix.ilm.eventmanager.AbstractEventHandler`

The Default Mission Domain model Subscribes to Phoenix Events of the type "Mission" which can indicate a mission is being prepped for, starting, or ending, as shown in Table 6. Relevant information such as the IOs that are related to the mission, the estimated amount of storage space needed, and the mission importance are extracted from the Mission Event and used to create the appropriate ILM Events including Group Update Events, Valuation Events, Cleanup Events, and NeedSpace Events.

```
eventType: "Mission"

initialize: Subscribes the ILM Service to receive Phoenix Events of type "Mission"

processEvent: Triggers the specified ILM Actions. See Table 6 for details.
```

Table 6. The Mission Events in the Default Mission Domain Model

Mission State	Description	Triggered ILM Events	Description of ILM Actions
MissionPrep	Preparing for an upcoming mission	*Cleanup*	sorts IOs on level 0 and 1
MissionBegin	Start of a mission	*GroupUpdate*	Creates a group matching the mission predicate
		Valuation	Triggers valuation over all IOs matching the mission predicate
		NeedSpace	Moves enough space for the mission
MissionEnd	End of a mission	*GroupUpdate*	Removes the group associated with this mission
		Valuation	Triggers valuation over all IOs that matched the mission predicate

4.3.4.2.1 Mission Context

Extends the GroupContext (described in Section 4.3.7.1), along with other Group Details. It also stores the Mission State and expected space needed.

Interface:	`mil.af.rl.phoenix.ilm.grouping.MissionContextInterface`
Implementation:	`mil.af.rl.phoenix.ilm.groups.MissionContext`
Extends:	`mil.af.rl.phoenix.ilm.groups.GroupContext`

4.3.4.2.2 Mission Event

A Phoenix Event with an event type of "`Mission`" and a `MissionContext` as the body.

Interface: `mil.af.rl.phoenix.event.events.MissionEventInterface`
Implementation: `mil.af.rl.phoenix.event.events.MissionEvent`
Extends: `mil.af.rl.phoenix.event.events.Event`

4.3.4.3 Policy Event Handler

Interface: `mil.af.rl.phoenix.ilm.eventmanager.IlmEventHandlerInterface`
Implementation: `mil.af.rl.phoenix.ilm.eventmanager.PolicyEventHandler`
Extends: `mil.af.rl.phoenix.ilm.eventmanager.AbstractEventHandler`

The Policy Event Handler Subscribes to Phoenix Events of the type "Policy" and provides a remote administration interface to the ILM. Policy Events can be used to add/remove or update event handlers, and remotely pass a list of ILM Events directly to the ILM Controller.

```
eventType: "ILM_Policy"

initialize: Subscribes the ILM Service to receive Phoenix Events of type "ILM_Policy"

processEvent: Updates the specified handler. If a Handler Name is specified and the
included Handler is null, it will delete the handler with the corresponding name.
Passes any included ILM Events to the controller.
```

4.3.4.3.1 ILM Policy Context

The ILM Policy Context (`mil.af.rl.phoenix.ilm.groups.IlmPolicyContext`) is the body of ILM Policy events. It extends the Phoenix Base Context (`mil.af.rl.phoenix.contexts.BaseContext`) and contains the fields specified in Table 7.

Table 7. Fields in the ILM Policy Context

Field Name	**Type**	**Description**
Handler Name	`String`	Name of the event handler to update
Event Handler	`IlmEventHandlerInterface`	New Event Handler (null means delete the named handler)
ILM Events	`LinkedList<ILMEventInterface>`	List of ILM Events to directly hand to controller

4.3.4.3.2 ILM Policy Event

A Phoenix event with type "`ILM_Policy`" and a body consisting of an `ILMPolicyContext`.

Interface: `mil.af.rl.phoenix.event.events.IlmPolicyEventInterface`
Implementation: `mil.af.rl.phoenix.event.events.IlmPolicyEvent`
Extends: `mil.af.rl.phoenix.event.events.Event`

4.3.4.4 ISQM Listener

Implementation: `mil.af.rl.phoenix.ilm.eventmanager.ISQMListener`
Interface: `mil.af.rl.phoenix.ilm.eventmanager.IlmEventHandlerInterface`

`mil.af.rl.phoenix.ilm.mockqed.PolicyChangeListener`

Extends: `mil.af.rl.phoenix.ilm.eventmanager.AbstractEventHandler`

The ISQM Listener connects to an ISQM service and translates QED style policies into ILM actions and groupings. Specifically, a QED policy is turned into a Group Context (described in 4.3.7.1) with *Importance* as a Stored Value. The ISQM Listener then issues a Group Update event, containing the new group context. We use a MockISQM for demonstration purposes so demonstration of VFILM does not depend on QED, and the current prototype requires the Mock-ISQM Service to be collocated.

```
eventType: null

initialize: Registers the event handler as a listener with the ISQM Service

processEvent: ISQM Listener does not react to Phoenix Events
```

When the ISQM Service issues a policy update, it notifies all subscribers via a call to

```
public void policyChanged()
```

The ISQM Listener then retrieves a list of all policies from the ISQM Service. If a policy applies to QUERY operations, we assume that it should affect the valuation of IOs (because valuation can affect query times based on IO storage location).

The policy is then translated into an appropriate ILM Group Context, which is then issued via a Group Update Event and Valuation Event. Table 8 shows an example translation.

Table 8. An Example Translation from a QED Policy to an ILM Group Context.

Incoming QED Policy		**Resulting ILM Group Context**	
Policy Name	#Valuation#BFT.policy	Group ID	QED#Valuation#BFT.policy
Conditions	condition.operations=QUERY condition.types = com.bbn.report.blueforcetrack	Predicate	/type='com.bbn.report.blueforcetrack'
Precedence	3	Precedence	3
Importance	4	Stored Values	Importance = 4
Preferences		Evaluation Rule	QED_Policy_FCL

4.3.4.5 Location Manager

Interface: `mil.af.rl.phoenix.ilm.eventmanager.IlmEventHandlerInterface`
`mil.af.rl.phoenix.ilm.CoordinateUpdatable`

Implementation: `mil.af.rl.phoenix.ilm.locationdemo.LocationManager`

Extends: `mil.af.rl.phoenix.ilm.eventmanager.AbstractEventHandler`

The Location Manager is a demonstration component that subscribes to IOs published by a specific publisher containing positional data. It then issues ILM Events that highly value all other IOs in the proximity of the tracked unit. We created the LocationManager mainly to demonstrate the wide range of ILM behaviors we could achieve with minimal changes to the system as a whole. Because it relies on positional data being in a certain form, it is best suited for demonstrations and inspiration for future event handlers.

```
eventType: null

initialize: Registers a predicate with the Information Brokering Service to receive
IOs from a given publisher. Spawns a Location Subscriber, which serves as the callback
for the registered predicate. The Location Subscriber passes the new coordinate infor-
mation from the IO subscription via the Coordinate Updatable Interface.

processEvent: Does not respond to Phoenix events.
```

4.3.5 Prototype Implementation of the ILM Controller

The *ILM Controller* (`mil.af.rl.phoenix.ilm.controller.Controller`) implements the ILM Controller Interface (`mil.af.rl.phoenix.ilm.controller.ControllerInterface`). The *ILM Controller* drives the behavior of the ILM in response to ILM Events. Currently there are seven defined events which all ILM actions are composed of. ILM Events can be handed to the ILM Controller as lists which will be executed in order, or as individual events. Multiple calls to the ILM Controller may be executed out of order. If the order in which events are executed is important, they must be passed to the ILM Controller in a list. For more details on ILM Events see Section 4.2.4.

4.3.6 Prototype Implementation of the Value Depreciation Function

The Value Depreciation Function (`mil.af.rl.phoenix.ilm.vdf.ValueDepreciationFunction`) implements the Value Depreciation Function Interface (`mil.af.rl.phoenix.ilm.vdf.ValueDepreciationFunctionInterface`). It maintains a mapping of evaluation rules (`Map<String, EvaluationRuleInterface>`) within a Value Depreciation Function Context (`mil.af.rl.phoenix.ilm.contexts.ValueDepreciationFunctionContext`) and uses these rules to calculate an IO's value. This rule mapping treats the key as an identifier for the rule, and the entry is the rule itself. As described previously, group membership plays a large role in the valuation process. An IO’s Group Context includes a field with the name of the specific valuation rule to use (corresponding to an entry in this rule map) and a set of possible input values to the valuation process.

IO valuations are carried out in batches asynchronously via an *EvaluateTask* (`mil.af.rl.phoenix.ilm.vdf.ValueDepreciationFunction.EvaluateTask`). An Evaluate Task accepts a list of IOs and a reference to an ILM Index containing the IOs. The Index is used to first determine a given IO's group membership and then to update the IO's Index entry.

4.3.6.1 FCLRule

All evaluation rules implement the EvaluationRuleInterface (`mil.af.rl.phoenix.ilm.eventmanager.EvaluationRuleInterface`). The VFILM prototype contains one such implementation, the FCLRule (`mil.af.rl.phoenix.ilm.vdf.FCLRule`), which uses Fuzzy Control Logic to calculate Information values. Each FCLRule contains a Fuzzy Inference System (FIS) and a mapping of fuzzy variables used for inputs, as shown in Figure 14. We use *jFuzzyLogic* to implement the FCLRule. jFuzzyLogic is an open source Java implementation of Fuzzy Control Logic available at http://jfuzzylogic.sourceforge.net.

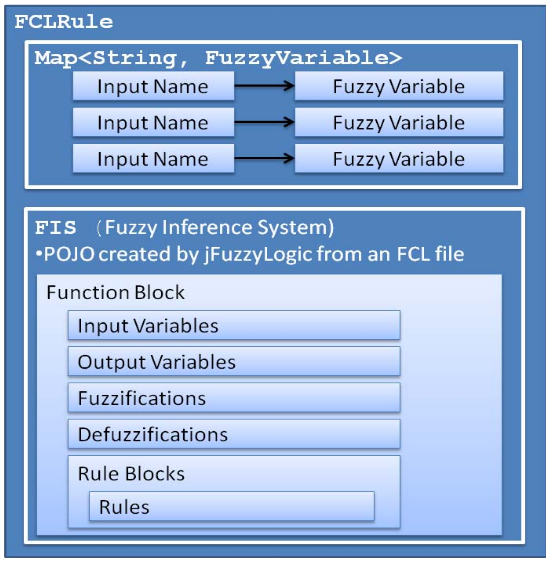


Figure 14. Layout of FCLRule

4.3.6.2 FIS

The FIS (fuzzy inference system) is a Java representation of an FCL program. It is created by jFuzzyLogic from a FCL file. Input variables are specified by the method

```
setVariable(String variableName, double variableValue);
```

Once all the inputs have been specified a call to

```
evaluate()
```

runs the inference system. Lastly a call to

```
getVariable("move").defuzzify();
```

defuzzifies and returns a numeric value for the output variable "move". This is the value we assign as an IO's value. How we specify input variables is described in Section 4.3.6.3 below. The specifics of an FCL file are described in Section 4.3.6.4.

4.3.6.3 Fuzzy Variables

A mapping of input names to fuzzy variables is used to determine the input set (`Map<String, FuzzyVariableInterface> variableMap`). The keys in the map are the names of the input variables, and each entry is a FuzzyVariable that returns the numeric value of the input. This value can be calculated based on properties of the IO, such as age or size, or values stored in its Group Context. The prototype implementations of VFILM's FuzzyVariables are described in Table 9.

Table 9. Fuzzy Variables prototyped in the VFILM prototype

Fuzzy Variable	Description
IOSizeVariable	returns an IO's payload size
AgeVariable	returns an IO's age
MissionVariable	returns a "Mission Importance" value for the IO's dominant group

The fuzzy variable interface includes one method:

```
public double calc(InformationInterface io, Map<String, Object> valueMap)
```

The value returned by this method will be the named variable's numeric input.

4.3.6.4 Fuzzy Control Logic

An FCL File contains one or more function blocks. Each function block specifies how values are passed in, processed, and returned. This is specified by five distinct sections [11]:

- Input variables.
- Output variables.
- The fuzzification of numeric inputs into various fuzzy sets.
- The defuzzification of fuzzy sets into numeric outputs.
- Rules that define how to combine various sets.

The details of each section follow.

4.3.6.4.1 Input and Output variables

Input variables are specified by the keyword *VAR_INPUT*. For example,

```
//Defines the name of input variables
VAR_INPUT
    ioSize : REAL;
    missionStatus : REAL;
    age : REAL;
END_VAR
```

specifies that there are three input variables, *ioSize*, *missionStatus*, and *age*, and that each variable is a real number.

Similarly output variables are specified by the keyword *VAR_OUTPUT*.

```
//Defines the name of output variables.
VAR_OUTPUT
    move : REAL;
END_VAR
```

specifies that there is one real output named *move*.

4.3.6.4.2 Rule Block

The behavior of a fuzzy algorithm is defined in one or more rule blocks with each rule block consisting of at least one rule. Suppose we want a rule that moves old, large, and irrelevant IOs. Our FCL file could include the following rule block:

```
RULEBLOCK first
    // Use 'min' for 'and' (also implicit use 'max'
    // for 'or' to fulfill DeMorgan's Law)
    AND : MIN;
    // Use 'min' activation method
    ACT : MIN;
    // Use 'max' accumulation method
    ACCU : MAX;

RULE 1 : IF ioSize IS large
         AND ioAge IS old
         AND ioRelevance IS minimal
         THEN move IS likely;

END_RULEBLOCK
```

The first line

```
RULEBLOCK first
```

assigns the name *first* to this rule block. The next line

```
    AND : MIN;
```

identifies the operator to use for an *AND*. The *AND* operator is clearly defined in binary logic. With fuzzy logic, the values being logically combined using an *AND* will have values between 0 and 1 (inclusive), so the meaning of *AND* must be defined. In this case, we are simply taking the minimum of the two values. Similarly, we are (implicitly) taking the max for an *OR*. The next line

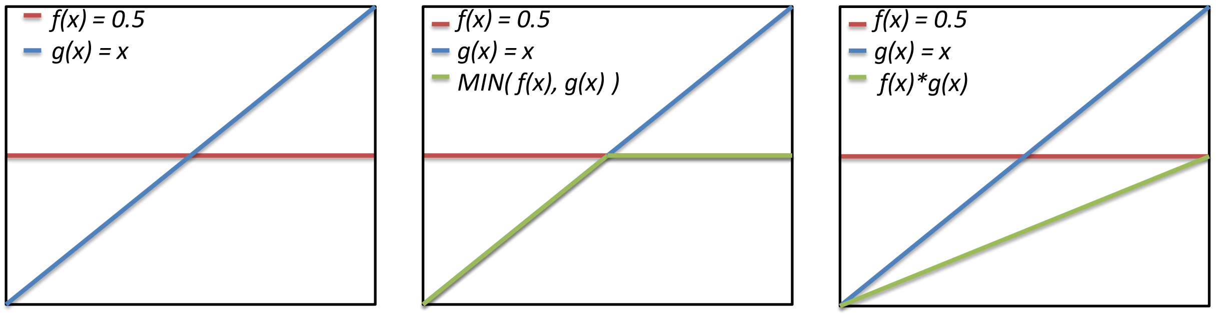
```
    ACT : MIN;
```

indicates the activation method. The activation method defines how the condition (the *IF* part of a rule statement) affects the consequence (the *THEN* part of a rule statement). Two possible activation methods are *MIN* and *PROD*. *MIN* takes the minimum of the condition and the consequence, while *PROD* takes the product of the condition and consequence.

To illustrate how the activation method works, consider the following rule

```
if f(x) then g(x);
```

In fuzzy logic, the result of this statement is *g(x)* affected by *f(x)* in the way specified by the activation method. This means that if the activation method is *MIN*, the result of the above rule is *MIN(f(x), g(x))*. If the activation method is *PROD*, the result of the above rule is *f(x)*g(x)*. This is illustrated in Figure 15 for *f(x)=0.5* and *g(x)=x*.



(a) The functions *f(x)* and *g(x)*. (b) *MIN(f(x), g(x))* (c) *PROD(f(x), g(x)*

Figure 15. Graphical representations of the MIN and PROD activation methods.

Notice that both of these activation methods work with binary sets. With binary logic *f(x) = 0* or *1*. If *f(x) = 0*, then *MIN(f(x),g(x)) = PROD(f(x),g(x)) = 0*. Likewise, if *f(x) = 1*, then *MIN(f(x),g(x)) = PROD(f(x),g(x)) = g(x)*. That is, if *f(x)* is false, the result is false, and if *f(x)* is true, the result is the consequence *g(x)*.

Figure 16 shows the use of the *MIN* activation method in VFILM, for an example situation in which the condition evaluates to a value of *0.5* and the consequence is *move IS unlikely*. The value of the condition (0.5) is the *y* axis value, 0.5, represented by the horizontal line in Figure 16. The consequence (*move IS unlikely*) is the full *unlikely* set. Therefore, the minimum of the condition and the consequence is the subset of *unlikely* under the membership value 0.5, shown by the shaded area in Figure 16.

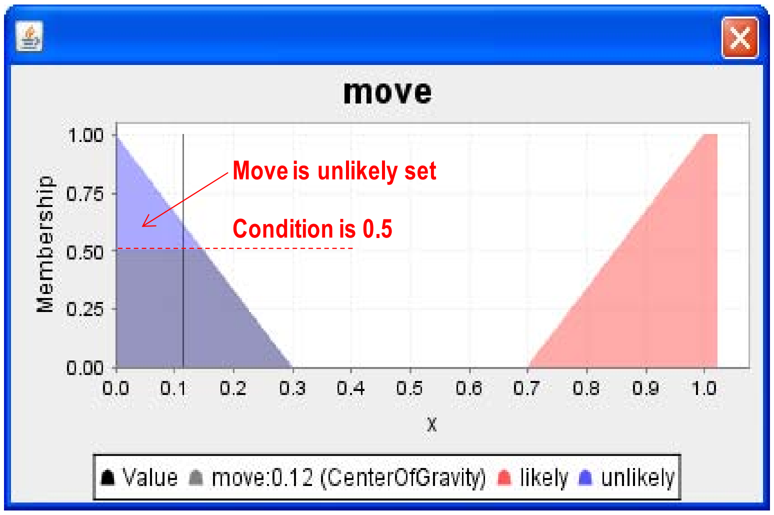


Figure 16 Use of MIN for an activation method, where the *condition* has a value of 0.5, and the consequence is *move IS unlikely*.

Figure 17 shows the same example, but using the *PROD* activation method. In this case, the result set contains the values of the *unlikely* set, multiplied by 0.5, essentially halving the set. The result is shown by the shaded area in Figure 17.

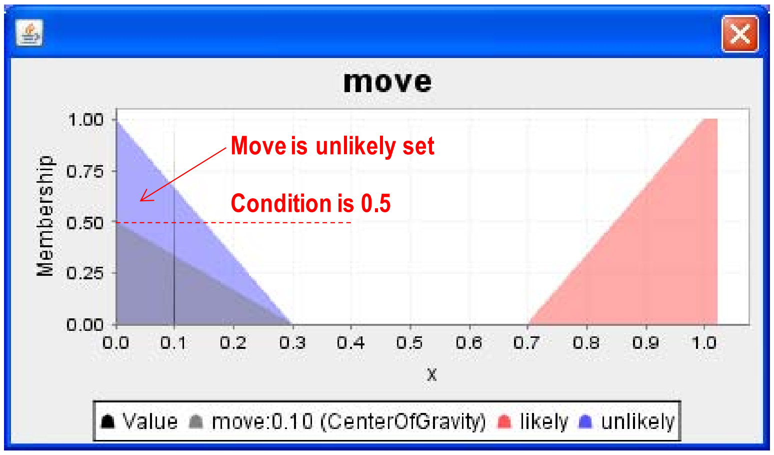


Figure 17. Use of PROD for an activation method, where the *condition* has a value of 0.5, and the consequence is *move IS unlikely*.

After that the accumulation method for each fuzzy set is defined.

```
    ACCU : MAX;
```

This is the method used when combining degrees of membership for the same set. For instance, if we have a set that states move is in the *likely* set with a value 0.5, and another that states move is in the *likely* set with a value of 0.75, we are deconflicting this by taking the max of the two.

Lastly, there are the listings of rules.

```
RULE 1 : IF ioSize IS large
         AND ioAge IS old
         AND ioRelevance IS minimal
         THEN move IS likely;
```

This rule calculates *move's* membership in the *likely* set based upon size, age, and relevance.

```
IF ioSize IS large
```

Will be a value between 0 and 1 based upon ioSize's membership in large,

```
         AND ioAge IS old
```

The *AND* indicates we are using the minimum of the previous value and the number representing *ioAge*'s membership in *old.*

```
         AND ioRelevance IS minimal
```

The *AND* once again indicates we are using the minimum of the previous value and *ioRelevance's* membership in *minimal.*

```
         THEN move IS likely;
```

states that *move* will be in the set *likely* with a value resulting from the *AND's* of the three conditions.

The degrees of membership of ioSize, ioAge, and ioRelevance, are all calculated by the *fuzzification* process (described in Section 4.3.6.4.3, next). The output of multiple rules is calculated via the *defuzzification* process (described in Section 4.3.6.4.4 below).

4.3.6.4.3 Fuzzification

As discussed previously descriptions such as *small, medium*, and *large* are not binary designation. Some things are very large, some are more medium, and others may be in between. To determine how large an IO is, we *fuzzify* its size, and see how far inside of the *large* set it falls. This is performed by the FCL *FUZZIFY* keyword, such as in the following code:

```
//Fuzzifies ioSize using a piece-wise linear function
FUZZIFY ioSize
    TERM small := (0,1)(100000,0);
    TERM large := (10000,0)(1000000,1);
END_FUZZIFY
```

The first line

```
FUZZIFY ioSize
```

indicates that the input variable *ioSize* will be fuzzified by this statement. The statement

```
TERM small := (0,1)(100000,0);
```

defines the fuzzy set *small* and indicates where an IO lies within the set based on the IO's size (specified by the variable *ioSize*). The first term of each parenthetical pair defines the bound of the set, i.e.,

```
TERM small := (0,1)(100000,0);
```

says that IOs with a size between 0 and 100,000 bytes lie somewhere in the small set. The second term of each parenthetical pair defines the membership value of elements in the set (i.e., where each IO lies in the set. In other words,

```
TERM small := (0,1)(100000,0);
```

indicates that IOs that are 0 bytes in size are completely in the small set (i.e., have a membership value of 1) and that IOs of size 100,000 bytes or more are completely not members of the set (i.e., have a membership value of 0). IOs of size between 0 and 100,000 bytes are partial members of the set, with their membership in the set decreasing linearly as their size increases.

Likewise, the statement

```
TERM large := (10000,0)(1000000,1);
```

defines the fuzzy set *large* as containing IOs with sizes between 10,000 bytes and 1,000,000 bytes (or more), with membership in the set increasing linearly as the size of the IO increases. IOs larger than 1,000,000 bytes are fully members of the *large* set, while IOs smaller than 0 bytes (if such a thing were possible) are fully members of the *small* set.

The syntax of the *TERM* statements in this code defines a piece-wise linear function for the sets, as shown in Figure 18. Other membership functions available as part of jFuzzyLogic are described in Section 4.3.6.4.5.

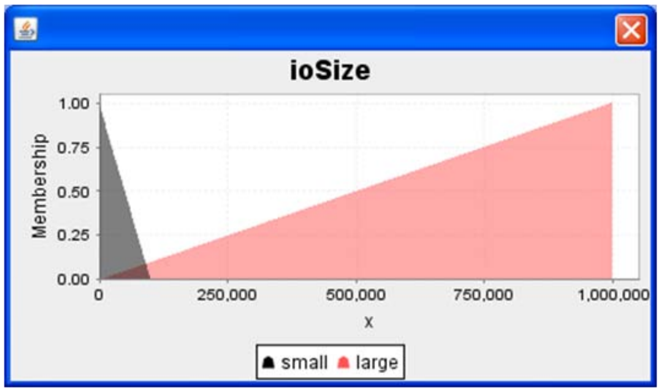


Figure 18. Piece-wise linear graphs for the membership functions of the *small* and *large* fuzzy sets.

4.3.6.4.4 Defuzzification

Defuzzification is the process by which degrees of membership are converted into a single numeric value. This process is specified by the FCL *DEFUZZIFY* keyword, such as in the following code.

```
//Defuzzifies the output variable move
DEFUZZIFY move
```

```
    TERM unlikely := (0,1) (0.3,0) ;
    TERM likely := (0.7,0) (1,1);
    METHOD : COG;
    DEFAULT := 0.25;
END_DEFUZZIFY
```

The first line

```
DEFUZZIFY move
```

identifies that the ouput variable *move* will be defuzzified by this statement. The next two lines

```
    TERM unlikely := (0,1) (0.3,0) ;
    TERM likely := (0.7,0) (1,1);
```

Define the membership functions associated with this output variable in the same manner as described in Section 4.3.6.4.3 and result in the fuzzy sets shown in Figure 19.

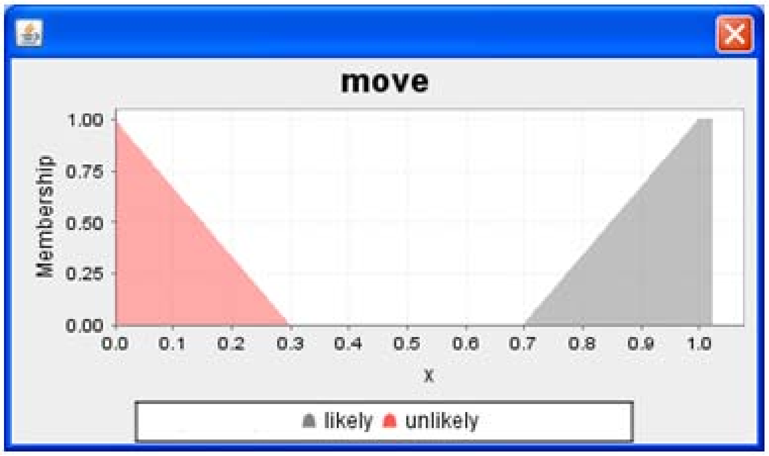


Figure 19. Piece-wise linear graphs for the membership functions of *move* is *likely* and *unlikely*.

The next line

```
METHOD : COG;
```

defines the method used to defuzzify the output variable's degrees of membership in its fuzzy sets. In this case, we utilized a center of gravity (COG) calculation to determine the output. For example, consider a case where *move's* degree of membership in un*likely* is ~ 0.7 and its degree of membership in *likely* is approximately 0.2. This is shown in Figure 20, where the shading in each set represents move's degree of membership in the set.

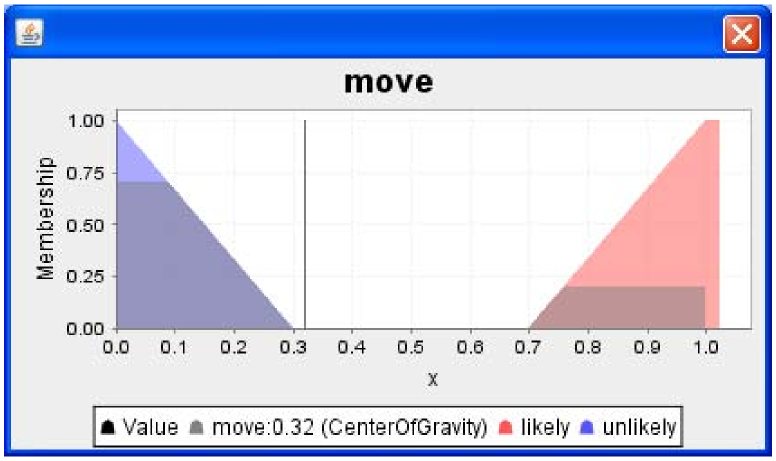


Figure 20. Example membership set where move's degree of membership in unlikely and likely is ~0.7 and ~0.2, respectively.

The center of gravity method takes the average of the two sets, weighted by the degree of membership in each set, and calculates a single numeric output value for *move.* The resulting value is represented by the vertical line in the graph in Figure 20, in this case a membership value of 0.32.

The next line

```
    DEFAULT := 0.25;
```

simply states a default return value in the event that the degree of membership in each set is 0.

4.3.6.4.5 Membership Functions

The examples in the previous sections showed piece-wise linear functions as membership functions. The following section describes other membership functions that jFuzzyLogic provides[1].

Piece-wise Linear

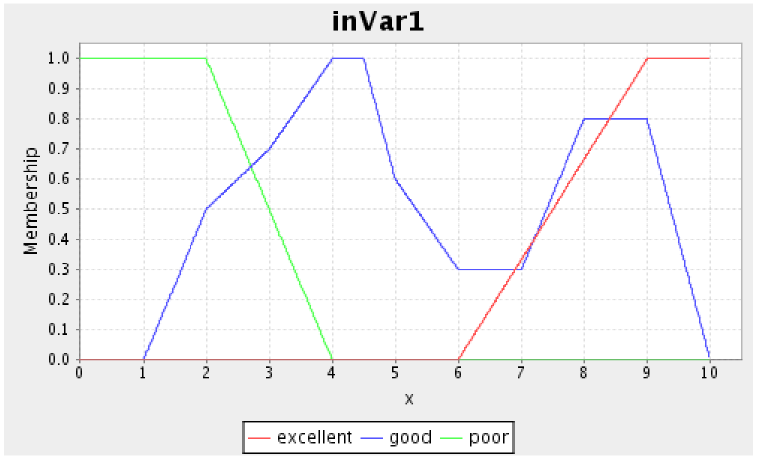


Figure 21. Example piece-wise linear membership functions

[1] These come from http://jfuzzylogic.sourceforge.net/html/membership.html.

```
   Usage: (x_1, y_1) (x_2, y_2) .... (x_n, y_n)
```

```
FUZZIFY inVar1
        TERM poor := (0,1) (2, 1) (4, 0) ;
        TERM good := (1, 0) (2, 0.5) (3, 0.7) (4,1) (4.5, 1)
                 (5, 0.6) (6, 0.3) (7, 0.3) (8, 0.8) (9, 0.8) (10,0);
        TERM excellent := (6, 0) (9, 1) (10,1);
END_FUZZIFY
```

Triangular

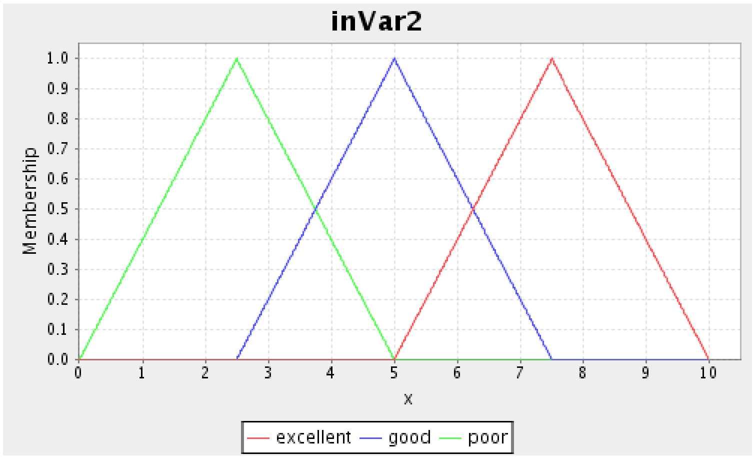


Figure 22. Example triangular membership functions

```
   Usage: trian min mid max
```

```
FUZZIFY inVar2
        TERM poor := trian 0 2.5 5;
        TERM good := trian 2.5 5 7.5;
        TERM excellent := trian 5 7.5 10;
END_FUZZIFY
```

Trapezoidal

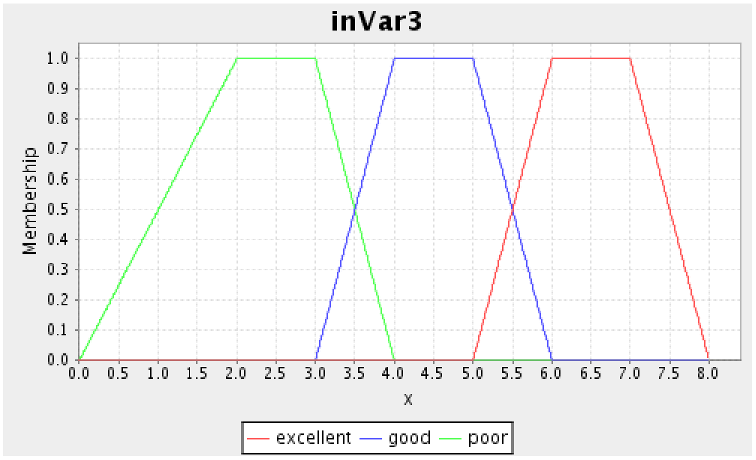


Figure 23. Example trapezoidal membership functions

```
Usage: trape min midLow midHigh max
```

```
FUZZIFY inVar3
        TERM poor := trape 0 2 3 4;
        TERM good := trape 3 4 5 6;
        TERM excellent := trape 5 6 7 8;
END_FUZZIFY
```

Gaussian

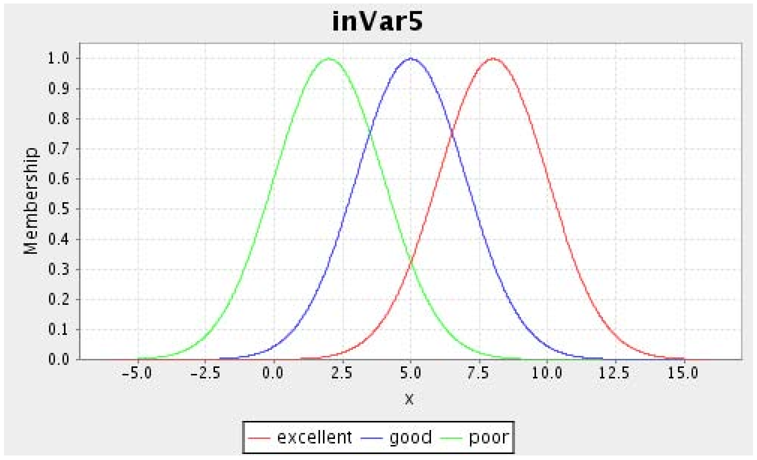


Figure 24. Example Gaussian membership functions

```
Usage: gauss mean stdev
```

```
FUZZIFY inVar5
        TERM poor := gauss 2 2;
        TERM good := gauss 5 2;
        TERM excellent := gauss 8 2;
END_FUZZIFY
```

Generalized bell

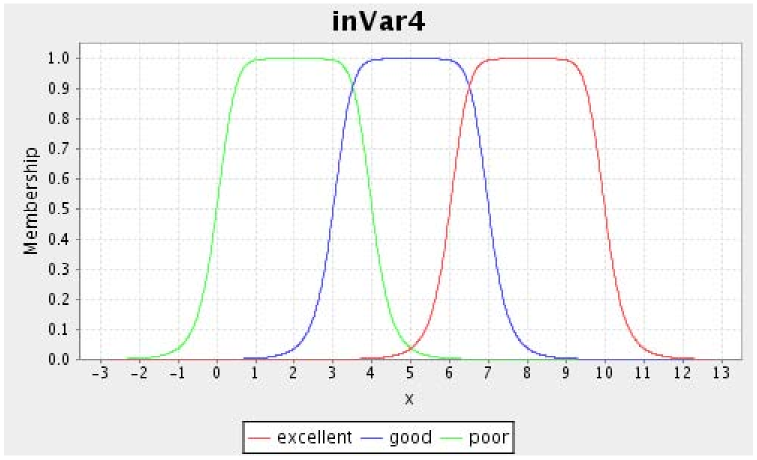


Figure 25. Example generalized bell membership functions

```
   Usage: gbell a b mean
```

```
FUZZIFY inVar4
        TERM poor := gbell 2 4 2;
        TERM good := gbell 2 4 5;
        TERM excellent := gbell 2 4 8;
END_FUZZIFY
```

Sigmoidal

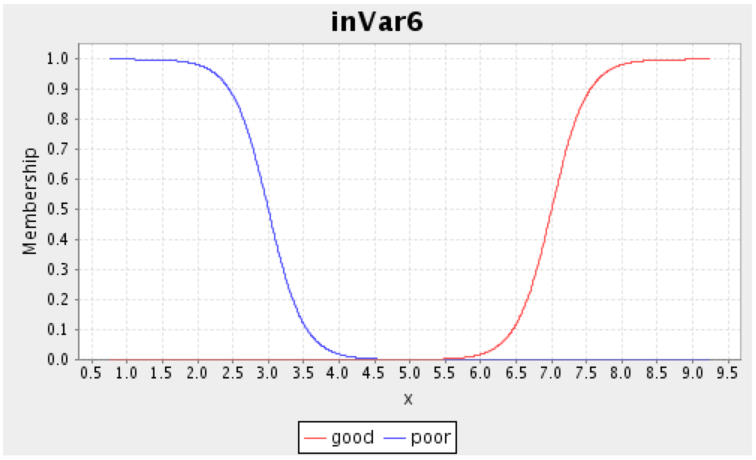


Figure 26. Example sigmoidal membership functions

```
   Usage: sigm gain center
```

```
FUZZIFY inVar6
        TERM poor := sigm -4 3;
        TERM good := sigm 4 7;
END_FUZZIFY
```

Singleton

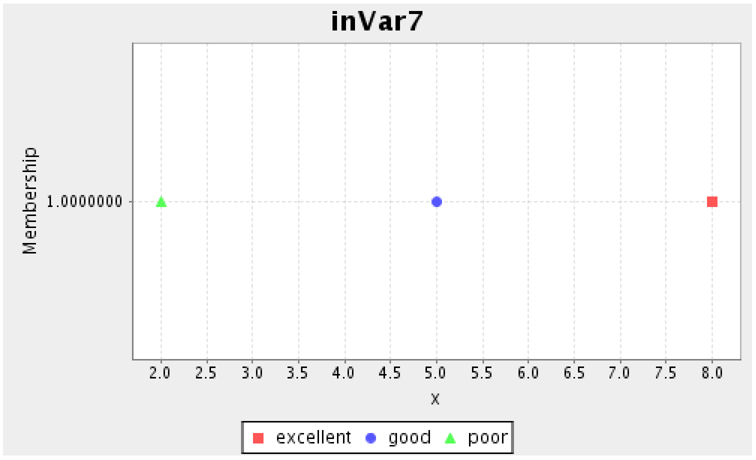


Figure 27. Example singleton membership functions

Usage: *X* (indicating the constant at which the variable has membership of 1)

```
FUZZIFY inVar7
        TERM poor := 2;
        TERM good := 5;
        TERM excellent := 8;
END_FUZZIFY
```

4.3.6.4.6 FCL Rule Example

The following section shows an example of how the valuation of an Information Object is performed with an example *FCLRule*. The example *FCLRule* uses the Variable Map specified in Table 10 and the *FIS* created from the FCL File specified in the following code listing.

Table 10. Mapping from Fuzzy Input Variable Names to Fuzzy Variables

Input Variable Name	**Fuzzy Variable Used**
"ioSize"	IOSizeVariable
"age"	AgeVariable
"missionStatus"	MissionVariable

Below is a full listing of the FCL File:

```
FUNCTION_BLOCK moveBlock

//Defines the name of input variables
VAR_INPUT
    ioSize : REAL;
    missionStatus : REAL;
    age : REAL;
END_VAR

//Defines the name of output variables.
VAR_OUTPUT
    move : REAL;
END_VAR

//Fuzzifies missionStatus using a linear stepwise function
FUZZIFY missionStatus
    TERM active := (0.0,0)(1.0,1);
END_FUZZIFY

//Fuzzifies age input using a sigmoidal function
FUZZIFY age
    TERM young := sigm -0.0001 60000;
    TERM old := sigm  0.00001 500000;
END_FUZZIFY

//Fuzzifies ioSize using a linear stepwise function
FUZZIFY ioSize
    TERM small := (0,1)(100000,0);
    TERM large := (10000,0)(1000000,1);
END_FUZZIFY

//Defuzzifies the output variable move
DEFUZZIFY move
    TERM unlikely := (0,1) (0.3,0) ;
    TERM likely := (0.7,0) (1,1);
    METHOD : COG;
    DEFAULT := 0.25;
END_DEFUZZIFY

RULEBLOCK first
    // Use 'min' for 'and' (also implicit use 'max'
    // for 'or' to fulfill DeMorgan's Law)
    AND : MIN;
    // Use 'min' activation method
    ACT : MIN;
    // Use 'max' accumulation method
    ACCU : MAX;

RULE 1 :   IF ioSize IS small
    THEN move IS unlikely WITH 0.7;

RULE 2 :   IF ioSize IS large
    THEN move IS likely WITH 0.5;

RULE 3 :   IF missionStatus IS active
    THEN move IS unlikely;

RULE 4:    IF missionStatus IS NOT active
    THEN move IS likely WITH 0.35;

RULE 5:    IF age IS young
    THEN move IS unlikely WITH 0.8;

RULE 6 :   IF age IS old
    THEN move IS likely WITH 0.4;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Consider an IO that is 90 KB, 8.33 minutes old, and belongs to a mission with priority of 0.7. First, the Fuzzy Variables extract these values and pass them to the FIS as inputs.

```
VAR_INPUT
    ioSize : REAL;
    missionStatus : REAL;
    age : REAL;
END_VAR
```

Next, each input is fuzzified,

```
//Fuzzifies missionStatus using a linear stepwise function
FUZZIFY missionStatus
    TERM active := (0.0,0)(1.0,1);
END_FUZZIFY
```

The mission priority (0.7) is mapped directly to mission status, as shown in Figure 28.

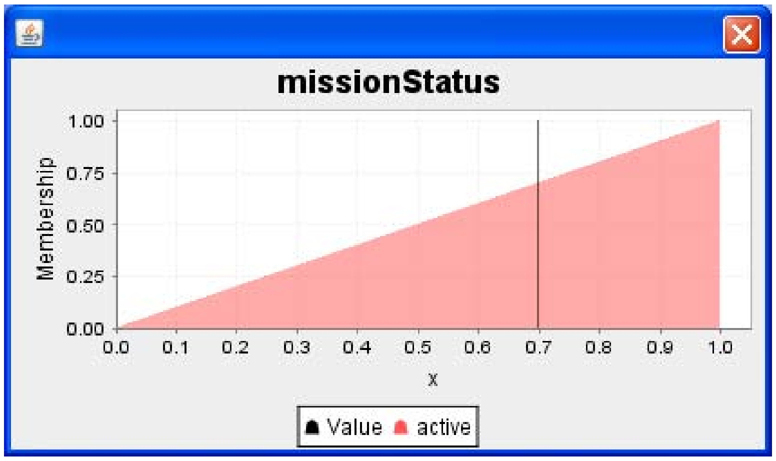


Figure 28. The missionStatus fuzzy set. The vertical line indicates the membership value of the IO in this set.

Next, we fuzzify the age input, this uses two sigmoidal functions to represent *young* and *old*.

```
//Fuzzifies age input using a sigmoidal function
FUZZIFY age
    TERM young := sigm -0.0001 60000;
    TERM old := sigm 0.00001 500000;
END_FUZZIFY
```

The IO's age of 8.33 minutes (500,000 ms) falls outside the *young* set (upper bound of 60,000 ms) and approximately half in the *old* group, as shown in Figure 29.

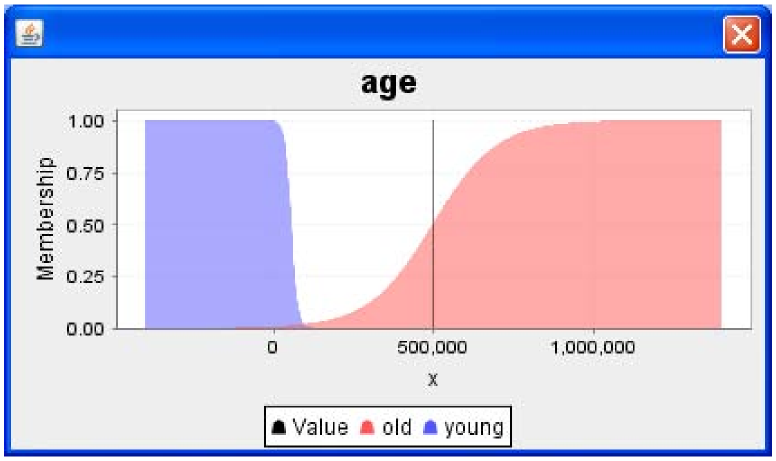


Figure 29. The age fuzzy sets. The vertical line indicates the membership value of the IO in the old set.

Next, we fuzzify IO size. The *small* and *large* sets are the same piece-wise sets used as a previous example, and shown in Figure 30.

```
//Fuzzifies ioSize using a linear stepwise function
FUZZIFY ioSize
```

```
    TERM small := (0,1)(100000,0);
    TERM large := (10000,0)(1000000,1);
END_FUZZIFY
```

With a size of 90 KB, the IO is about equally in the small and large sets.

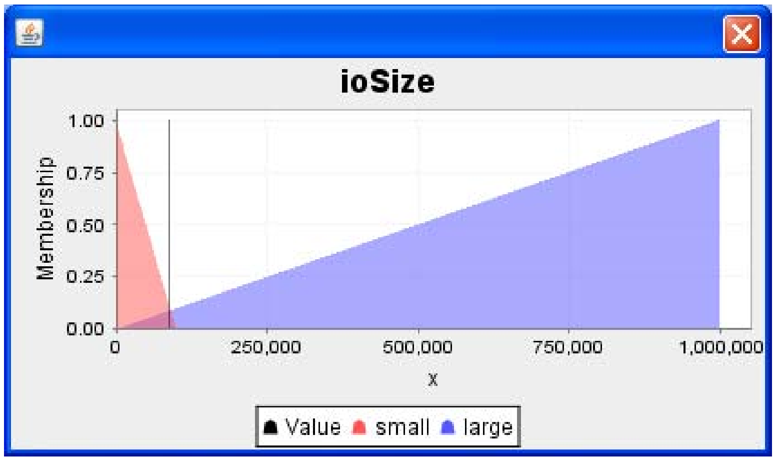


Figure 30. The size fuzzy sets. The vertical line indicates the membership value of the IO in the small and large sets.

Next, we illustrate how to defuzzify the *move* output set. The output value *move* provides a partial order of IO valuation, with respect to its likelihood for being moved or not. Therefore, we define *move* as two fuzzy output sets, *likely* and *unlikely*, shown in Figure 31. The numeric value of *move* is determined by performing a center of gravity (COG) calculation on the two associated sets. Additionally *move* is defined to have a default value of 0.25 in the event that it has no membership in *likely* and *unlikely*.

```
//Defuzzifies the output variable move
DEFUZZIFY move
    TERM unlikely := (0,1) (0.3,0) ;
    TERM likely := (0.7,0) (1,1);
    METHOD : COG;
    DEFAULT := 0.25;
END_DEFUZZIFY
```

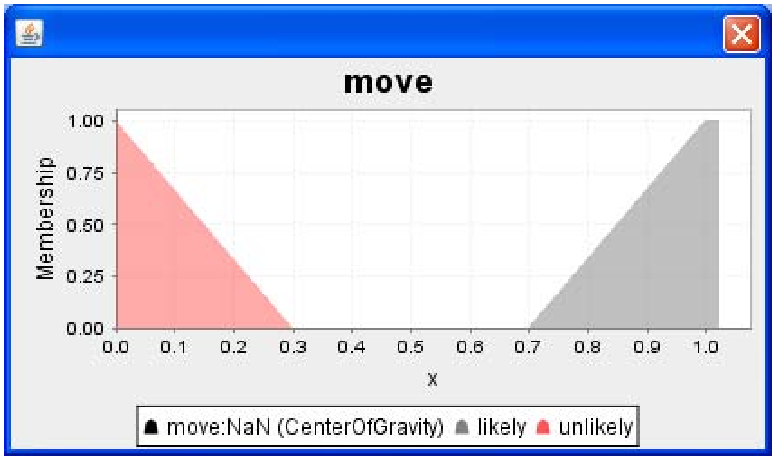


Figure 31. The move sets, likely and unlikely, defined around a center of gravity.

Once the inputs are fuzzified, the statements in the rule block are evaluated. Below, we describe how the membership of *move* in *likely* and *unlikely* is affected as each rule in the rule block is executed. The shaded section of each graph indicates the membership of *move* in each set and the vertical line indicates the resulting center of gravity calculation (i.e., the combined *move* value) at that point.

At each step we also indicate the resulting value of *move* if it was defuzzified at that point using the center of gravity calculation, this is represented by the vertical mark on the graph.

Before the rules can be executed, we need to define several operations. For details on these operations see Section 4.3.6.4.2.

```
    // Use 'min' for 'and' (also implicit use 'max'
    // for 'or' to fulfill DeMorgan's Law)
    AND : MIN;
    // Use 'min' activation method
    ACT : MIN;
    // Use 'max' accumulation method
    ACCU : MAX;
```

The first rule is:

```
RULE 1 : IF ioSize IS small
    THEN move IS unlikely WITH 0.7;
```

Because ioSize is slightly in the small set, this initially places the membership of *move* in *unlikely*, as shown in Figure 32.

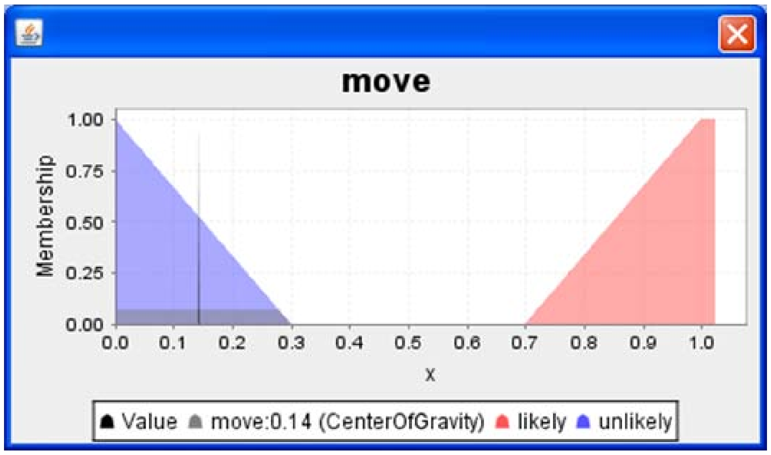


Figure 32. The result of evaluating Rule 1.

```
RULE 2 : IF ioSize IS large
    THEN move IS likely WITH 0.5;
```

Similarly, because ioSize is also slightly in the large set, this results in *move* having a small degree of membership in *likely*, as shown in Figure 33.

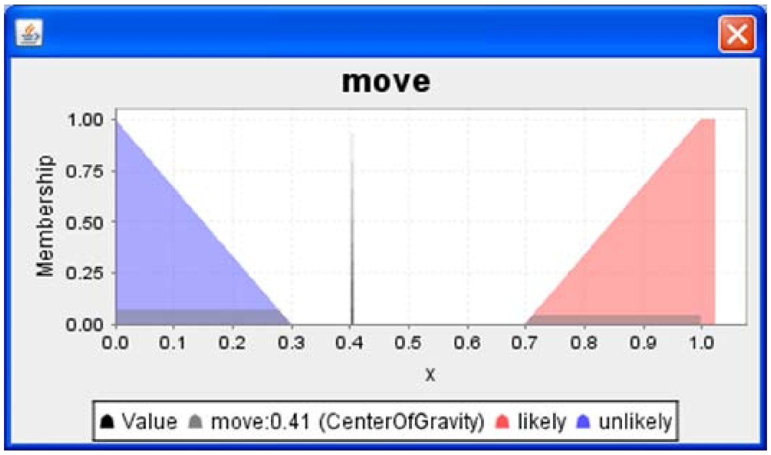


Figure 33. The result of evaluating Rule 2.

```
RULE 3 : IF missionStatus IS active
    THEN move IS unlikely;
```

missionStatus has a degree of membership of 0.7 in the *active* set, causing this rule to give move a degree of membership of 0.7 in the *unlikely* set. This membership in move is accumulated with the previous membership as defined in the rule block,

```
    // Use 'max' accumulation method
    ACCU : MAX;
```

Therefore we take the max of the two and *move*'s degree of membership in *unlikely* is now 0.7 (shown by the shading in the *unlikely* set in Figure 34).

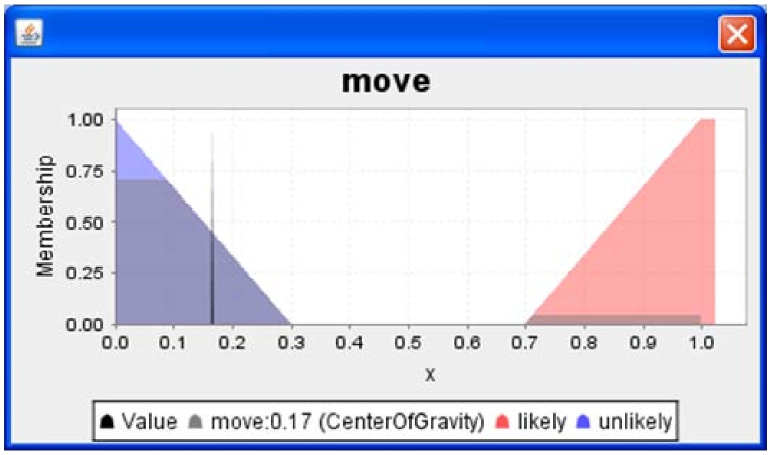


Figure 34. The result of evaluating Rule 3.

```
RULE 4: IF missionStatus IS NOT active
    THEN move IS likely WITH 0.35;
```

missionStatus has a membership of 0.7 in the *active* set (Figure 28), and therefore is slightly NOT active (0.3). However this rule is only given a weight of 0.35, and therefore the degree of membership of *move* in *likely* is calculated as NOT active * rule weight = 0.3 * 0.35 = 0.105. This is more than the previous calculated degree of membership in *likely* so our accumulation method assigns the new value as the degree of membership in *likely*, as shown in Figure 35.

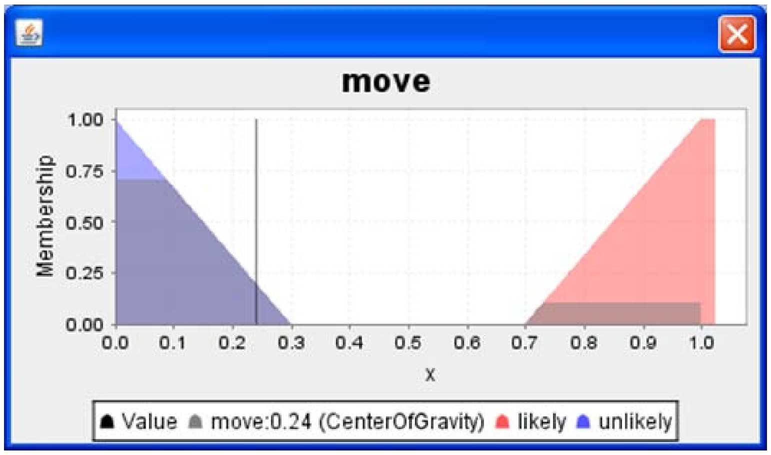


Figure 35. The result of evaluating Rule 4.

```
RULE 5: IF age IS young
    THEN move IS unlikely WITH 0.8;
```

Age is entirely not in the *young* set and therefore this rule has no effect, shown in Figure 36.

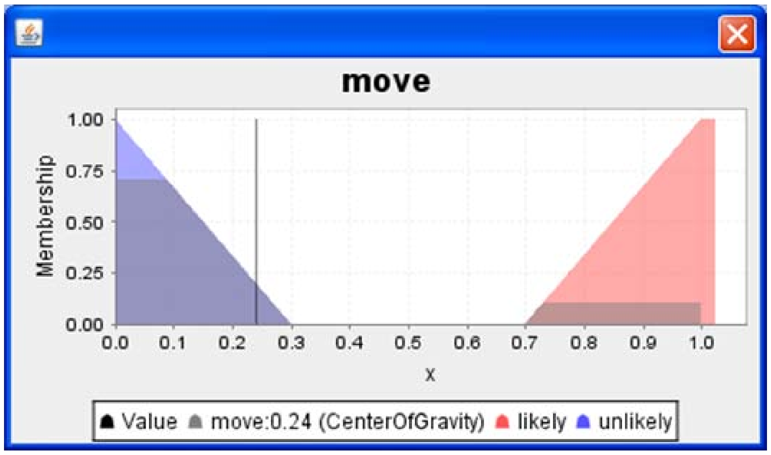


Figure 36. The result of evaluating Rule 5.

```
RULE 6 : IF age IS old
    THEN move IS likely WITH 0.4;
```

age has a degree of membership in the *old* set of approximately 0.5 (Figure 29), therefore this rule places *move* approximately half in the *likely* set, however the rule is only weighted at 0.4 and therefore the membership in the *move* set is closer to 0.2. The new degree of membership for *move* in *likely* is accumulated by taking the max with the previous value resulting in the degrees of membership seen in Figure 37.

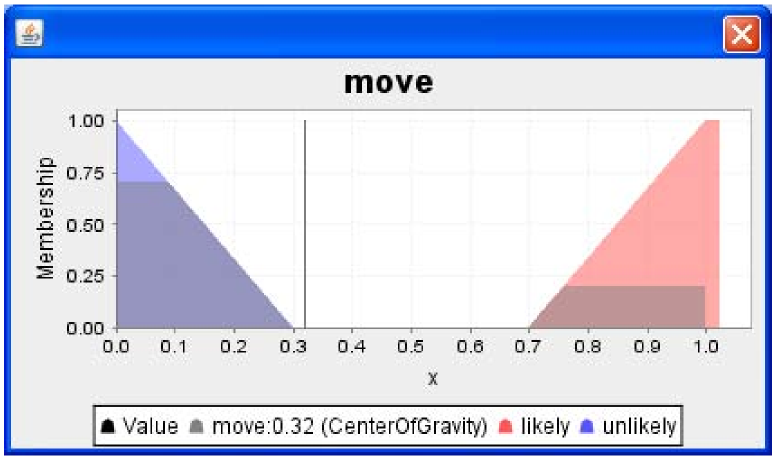


Figure 37. The result of evaluating Rule 6, the final rule. The vertical line indicates the output value of *move.*

As shown in Figure 37, the final values for *move* put it at approximately 0.7 in the *unlikely* set and approximately 0.2 in the *likely* set. The COG function creates a final *move* value of 0.32.

4.3.7 Prototype Implementation of the Group Manager

The Group Manager maintains a *Group Context Map* containing definitions of groups of IOs, as well as an *Accessor Map* that is used to index IOs and determine group membership.

The Group Manager is found in `mil.af.rl.phoenix.ilm.groups.GroupManager` and implements the interface `mil.af.rl.phoenix.ilm.groups.GroupManagerInterface.`

4.3.7.1 Group Context

Group contexts (`mil.af.rl.phoenix.ilm.groups.GroupContext`) are used as definitions for groups of Information Objects. They extend the Phoenix Base Context (`mil.af.rl.phoenix.contexts.BaseContext`) and implement the Group Context Interface (`mil.af.rl.phoenix.ilm.grouping.GroupContextInterface`). The Group Manager implementation stores these contexts in `Map<String, GroupContextInterface> groupMap` where the key is the group identifier. Each group context contains the five specific fields described in Table 11.

Table 11. The Fields in the Group Context

Name	Type	Description
Group Identifier	`String`	A unique identifier for the group.
Predicate	`String`	Group membership is determined by an XPath query.
Valuation Rule Name	`String`	The *ValueDepreciationFunction* stores multiple valuation rules in its rule map (`Map<String, EvaluationRuleInterface>`). The name of the rule that should be used for IOs in the group it is specified in this field. If no rule name is specified, the rule named "DEFAULT" will be used.
Stored Value Map	`Map <String, Object>`	A mapping of *String* to *Object* that contains additional group details. During the valuation process, this is passed to the *EvaluationRule* along with the IO being evaluated. A *FuzzyVariable* is able to use values in the map as inputs to the value calculation. For example, a group representing a mission with an importance of 0.7 would have an entry of [*Mission, 0.*7] in its Stored Value map. This 0.7 would be read as an input for any valuation rule that utilizes a *MissionVariable*.
Precedence	`int`	If an IO is a member of more than one group, the group with the highest precedence is used for valuation purposes. If multiple groups are tied for the highest precedence, then the IO is valued with respect to each group, and the IO is assigned the maximum value.

4.3.7.2 Accessor Interface

The *Accessor Interface* (`mil.af.rl.phoenix.ilm.AccessorInterface`) declares a single method that extracts the field for an IO's index entry from the IO.

```
public String getField(InformationInterface io)
```

The Group Manager maintains a mapping of these Accessors in `Map<String, AccessorInterface> accessorMap`, which is central to building indices for group management in the VFILM prototype. Group membership is based upon an IO's entry in the *ILMIndex*. When an IO is being indexed, the *ILMIndex* retrieves the *AccessorMap* from the *GroupManager*. Each entry contains a field with the name of each key in the *AccessorMap*. The contents of each field are determined by the corresponding A*ccessor*.

For example, consider the mapping of

“type” → *InfoTypeAccessor*

InfoTypeAccessor is an accessor that returns the Information Type of a given IO. Given an IO of type “mil.n.ship” this mapping would result in the following field in the index entry for that IO.

```
<type> mil.n.ship </type>
```

These index entries facilitate the movement and valuation of entire groups of IOs at a time. In the current prototype `AccessorInterface` is implemented by the six classes described in Table 12.

Table 12. Accessor classes implemented in the Group Manager prototype

Class Name	Description
ContextIDAccessor	Returns an IO's context ID
InfoTypeAccessor	Returns an IO's Information type
PublisherIDAccessor	Returns the “publisherId” attribute from an IO's context
PublishTimeAccessor	Returns the “publishTime” from an IO's context
ContextAccessor	Returns a specified attribute from an IO's context. The attribute is specified as a bean property: `String attr`
MetadataAccessor	Returns the results of a specified XPath query over the IO's metadata. The query is specified as a bean property: `String query`

4.3.8 Prototype Implementation of the ILM-HSM Adapter

Our prototype ILM-HSM Adapter is designed to work in a setting where the Repository Service is using Berkeley Repositories, there are two storage locations available (level 0 and level 1), and there is either not an active HSM or the HSM is only managing files located in level 1. This results in the ILM-HSM Adapter being responsible for all movement of information between level 0 and level 1+.

Additionally, the Berkeley Repo Adapter is responsible for storing the values and index entries of IOs that have been inserted into the repository. This is accomplished through the use of a Berkeley DB Value Store and an ILM Index respectively.

The Berkeley Repo Adapter (`mil.af.rl.phoenix.ilm.adapter.BerkeleyRepoAdapter`) implements the Adapter Interface (`mil.af.rl.phoenix.ilm.adapter.AdapterInterface`).

4.3.8.1 The ILM Index

The ILM Index (`mil.af.rl.phoenix.ilm.index.IlmIndex`) implements the ILM Index Interface (`mil.af.rl.phoenix.ilm.index.IlmIndexInterface`). It utilizes the Berkeley XML DB for Indexing IOs over fields relevant to grouping. This allows those IOs to be retrieved quickly when a group has to be updated or reevaluated. The Index works closely with the Group Manager and uses the Group Manager's *Accessor Map* to build its entries.

An IO is inserted into the Index via a call to

```
public void indexIO(InformationInterface io)
```

The ILM Index is used to retrieve the context ID's of IOs associated with a group via a call to

```
public List<String> getIds(String predicate)
```

Similarly, given a group of IOs, the groups they are associated with can be returned via a call to

```
public Map<String,List<GroupContextInterface>>
getGroupStatus( List<InformationInterface> ioList)
```

This returns a mapping of Context ID to a list of groups of associated IOs.

4.3.8.2 The Value Store

To determine which IOs need to be moved, the Berkeley Repo Adapter maintains a *Value Store* that tracks each IO's context ID, value, and storage level. Value Stores must implement the Value Store Interface (`mil.af.rl.phoenix.ilm.adapter.ValueStoreInterface`). The VFILM prototype provides two implementations:

- The Berkeley DB Value, `mil.af.rl.phoenix.ilm.adapter.values.Berkeley-DBValueStore`
- The In Memory Value Store, `mil.af.rl.phoenix.ilm.adapter.values.InMemory-ValueStore`.

The Berkeley DB Value store uses a Berkeley Database to store IO values. It is persistent between runs and is the recommended current implementation. The In Memory Value Store simply uses Java data structures (*LinkedLists* and *HashMaps*) to store Information values. Because of this, it is not persistent between runs and is likely to perform poorly on very large data sets. It was created primarily for testing and as part of an early version of the prototype.

Values are passed in and out of the Value Store as two different types of objects:

- `mil.af.rl.phoenix.ilm.iorep.IoRankingInterface` : Contains an IO's value and ID
- `mil.af.rl.phoenix.ilm.iorep.IoReferenceInterface` : Contains an IO's value, ID, and storage level.

For instance a call to

```
public void updateIOValue(List<IoRankingInterface> ioValueList, boolean areNew);
```

updates the values for all IOs in the list. If `areNew` is *true* then the values and ranks will be added to the Value Store with the assumption that they are located on level 0.

```
public void updateFuture(final Future<List<IoRankingInterface>> f, final boolean
areNew);
```

updateFuture is an asynchronous version of the previous method which accepts a Future object containing what will be the results of a call to the VDF.

```
public LinkedList<IoRankingInterface> getMoveList(int numToReturn, int level);
```

getMoveList returns a list of the IOs with the highest VDF valuation (i.e., the most depreciated, least important IOs) on the specified storage level.

```
public void move(List<IoRankingInterface> haveMoved, int newLevel);
```

The function *move* indicates that the specified IOs have moved to a new storage level.

```
public List<LinkedList<IoRankingInterface>> getOverLapLists();
```

getOverLapLists contains lists of IOs that are out of order on each storage level. For example, this list would contain all IOs on level 0 with a VDF value greater than any IO on level 1 (indicating that the IOs on level 0 are more depreciated than the ones on level 1). Similarly, every IO on level 1 with a value less than any IO on level 0 would be returned. These IOs can be thought of as out of order and are candidates to be moved during a *Clean Up* event

4.3.8.3 Interaction with the Berkeley Repository

The Berkeley Repo Adapter interacts with a Berkeley Repository via calls to the ILM Compatible Repository Service. The repository service can maintain multiple repository instances and manages them via UIDs. Upon initialization, the adapter stores the UID of the repository it corresponds to and includes this UID during calls to the repository service so that the correct repository is affected.

4.4 VFILM Demonstration

We developed a demonstration to showcase many of the features of the VFILM Prototype, including mission aware information valuation and movement, information grouping, monitoring of storage, runtime administration, custom query behavior, and integration with other policy systems (specifically, QED policy). This section provides an overview of the demonstration. A full description of how to build the VFILM prototype and run the demonstration is in the *VFILM Installation, Operations, Administration, & Demonstration Guide* [3].

The demonstration consists of three missions being carried out in a shared geographic region.

Mission A consists of Unit 1 traversing the area in a grid-like pattern continuously publishing small and large IOs.

Mission B involves Unit 2 traversing a parabolic path across the region. A custom event handler, the Location Manager, tracks Unit 2 and sends ILM Events to the ILM Controller making the IOs near Unit 2 highly important to the mission.

Mission C involves Unit 3 travelling to an area of interest. Once it reaches the area, it begins publishing 'recon' type IOs. A Mock ISQM Service issues a new QED Policy which makes IOs of this type important to the mission. An ISQM Listener Event Handler translates this policy into an ILM Group Update Event.

Three GUIs display what is occurring: the Location Demo GUI, the ILM Value Histogram, and the ILM Freespace Chart.

The Location Demo GUI, shown in Figure 38, displays a Cartesian grid representing the mission area. IOs that contain positional information in their metadata are plotted on the grid, and color coded to represent where they are stored. Red marks indicate the IO metadata and payload are located in the level 0 store. Blue marks indicate the IO metadata is located in the level 0

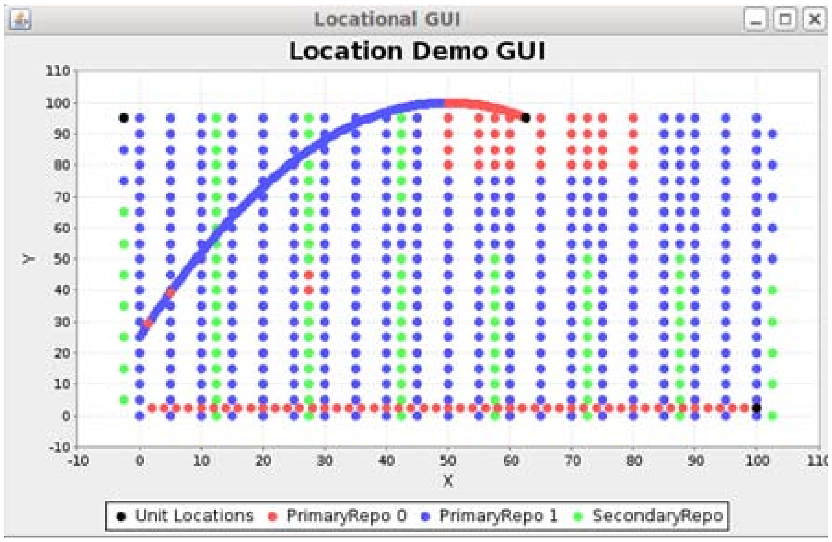


Figure 38. Location Demo GUI

store, but the payload is located in level 1. Green marks indicate the IO has been moved to a secondary repository and both the metadata and payload are in level 1 store. Black marks do not represent a storage location, but rather indicate that this is the most recent IO from a given publisher and therefore represents the publisher's last known location.

The ILM Value Histogram displays where IOs of different values are stored, as shown in Figure 39. The *x* axis indicates the value of the IO, with further to the left indicating it is more important to the ongoing scenario (and therefore has a lower *move* valuation), and the *y* axis indicates number of IOs with that value. As with the Location Demo GUI, the color represents storage location, red indicates metadata and payload are on level 0, blue indicates metadata is on level 0 and payload is on level 1, and green indicates metadata and payload are on level 1.

The ILM Freespace Chart, shown in Figure 40, displays the amount of free space on level 0 (*y* axis) over time (*x* axis). This is displayed as a red line. The current storage thresholds are

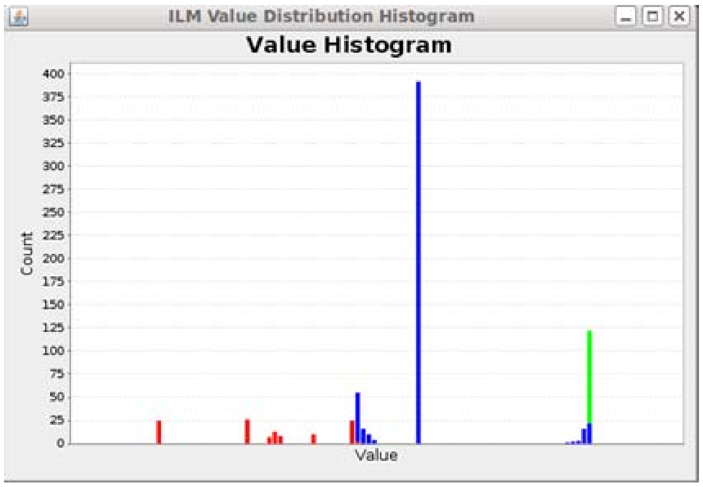


Figure 39. Value Histogram from VFILM Demonstration

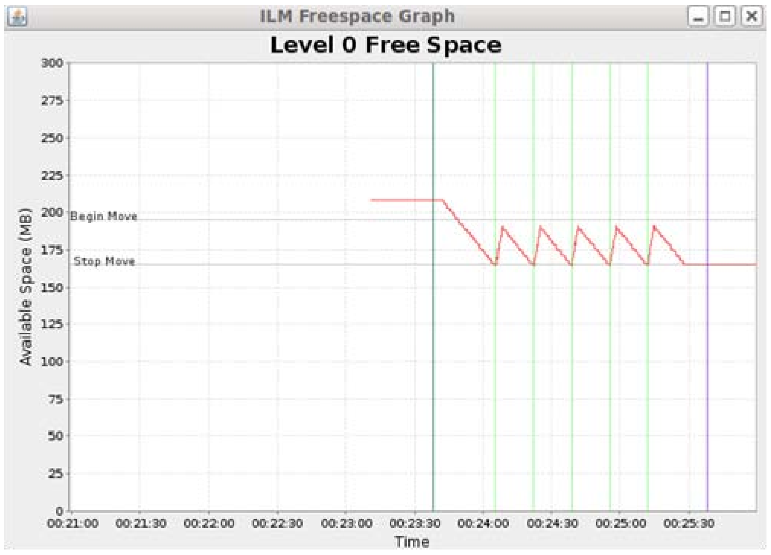


Figure 40. Free Space GUI from VFILM Demonstration

represented as gray horizontal lines. Vertical lines of various colors indicate ILM events that were sent to the controller with the value on the *x* axis indicating the time they were sent.

The demonstration includes the following actions:

- Mission A Starts, Unit 1 repeatedly publishes 3 small IOs, followed by 1 large IO.
- Mission A ends and IOs associated with it are devalued.
- The ILM's storage threshold is changed via an ILM Policy Event.
- The default evaluation rule is changed to more heavily weight age via an ILM Policy Event. IOs are revalued.
- The ISQM Listener is added as an Event Handler via an ILM Policy Event.
- Mission B Starts. Unit 2 travels in a parabolic course publishing IOs of type *bft* (i.e., blue force track). The Location Manager tracks Unit 2 and highly values all IOs in the region around it.
- An ISQM Policy is issued (by pushing the ISQM Trigger Button) and the ISQM Listener triggers a group update making 'recon' type IOs important.
- Mission C Starts. IOs in an area of interest become important.
- Unit 3 begins to travel to the area of interest. Unit 2 is still publishing IOs.
- Unit 3 approaches the area of interest and begins publishing 'recon' type IOs.
- Mission C ends. The area of interest is devalued, but the 'recon' IOs are still important. Unit 2 continues to publish IOs.
- Metadata Movement is triggered via an ILM Event. 100 IOs (metadata and payload) are moved to a secondary store.
- A query is issued for all IOs of type *bft* using a Phoenix Query Context. The *n* IOs are returned.
- A query is issued using an ILM Query Context. It specifies to only run the query over metadata stored in level 0 and only return IOs with payloads stored on level 0; *i* IOs are returned.

- A query is issued using an ILM Query Context. It specifies to only run the query over metadata stored in level 0 and only return IOs with payloads stored on level 1; *j* IOs are returned.
- A query is issued using an ILM Query Context. It specifies to only run the query over metadata stored in level 1 and only return IOs with payloads stored on level 0. No repositories are configured to stored data in this configuration and 0 IOs are returned.
- A query is issued using an ILM Query Context. It specifies to only run the query over metadata stored in level 1 and only return IOs with payloads stored on level 1; 100 IOs are returned.
- The total number of IOs returned is $n = i + j + 100$.

4.5 VFILM Experiment Results

This section describes the results of conducting the experiments described in Section 3.7.2 and collecting the metrics described in Section 3.7.1. More details about the experiments and results are in the *VFILM Experiment Results* document [16]. All of the experiments were executed during Spiral 2 of the VFILM project on the Spiral 1 prototype (the Spiral 2 prototype was delivered at the end of the VFILM contract). Since we made several functional and performance improvements to the VFILM prototype, the results documented in this report might actually be improved if they were re-run on the Spiral 2 prototype.

To facilitate re-running the experiments, we wrote the experiments as JUnit tests. Therefore, they are useful as functional and/or regression tests.

As mentioned in Section 3.7.1, we divided the metrics and therefore the tests into the following two sets:

- Functionality (Efficacy) – Tests and metrics that evaluate the VFILM software's ability to perform information lifecycle management and hierarchical storage management.
- Performance (Efficiency) – Experiments and metrics that evaluate the overhead, speed, and resource usage of the VFILM software while performing information lifecycle management and hierarchical storage management.

4.5.1 Summary of Results

The metrics being gathered in each category, the tests and experiments conducted to gather them, and a summary of results are described in Table 13 and Table 14.

Table 13. Functional Metrics, Tests, and Results

Metric number	Description	Tested By	Section	Result
F1	Responsiveness to events	MissionTriggersVDFTest	4.5.2	VDF valuation is triggered by appropriate mission events.
		MoveScalingTest	4.5.2	Appropriate mission events can trigger the HSM to move information objects.
		FreespaceTriggersMoveTest	4.5.2	Appropriate system events can trigger the HSM to move information objects.
F2	Repository maintenance	FreespaceTriggersMoveTest	4.5.3	VFILM maintains free space between defined maximum and minimum thresholds.
			4.5.3	The ILM is able to monitor the amount of free space in level 0.
F3	Publication	PublishPerformanceTest	4.5.4	The publish operation with the archive bit set re-

Metric number	Description	Tested By	Section	Result
	correctness			sults in the same number of archived IOs in the baseline Phoenix and in the VFILM system.
F4	Query correctness	QueryPerformanceTest	4.5.5	The baseline Phoenix and VFILM systems produce the same result set.

Table 14. Performance Metrics, Experiments, and Results

Metric number	Description	Experiment	Section	Result
P1	VDF scalability	VDFScalingTest. Measure the time to execute the VDF and compare to the number of IOs and the IO size.	4.5.6	Execution time of the VDF is *O(n)*, where *n* is the number of IOs being evaluated. Execution time of the VDF does not correlate to IO size.
P2	HSM scalability	MoveScalingTest. Time to execute HSM as the number of IOs increase.	4.5.7	Execution time of the HSM is affected by both the number of IOs moved and the size of the IOs. Larger IOs take longer because more data needs to be moved in the file system. More IOs take longer to move due to the filesystem operations of locating more files on disk.
		MoveScalingTest. Time to execute HSM as the total number of bytes increase.	4.5.7	
P3	Publication performance	PublishPerformanceTest. Time to complete archive for published IOs.	4.5.8	Publishing with VFILM takes approximately the same time as the Phoenix baseline. Experiments showed a mean 5% reduction in average time to publish with archive using VFILM, only 1.5 standard deviations of the baseline Phoenix mean. VFILM used an average of 13% more CPU, however, due to additional services. This is over 5 standard deviations. We conclude that any latency overhead imposed by VFILM (none in these experiments) is statistically insignificant, but the CPU overhead is statistically significant. The number of services in the Spiral 2 prototype has been reduced, so the CPU overhead could potentially be lower with the final delivered prototype.
P4	Query performance	QueryPerformanceTest. Time to return queried IOs.	4.5.9	Experiments showed a mean 1.2% additional time on average to execute a query in VFILM versus the baseline Phoenix., less than 1.5 standard deviations of the baseline Phoenix mean. The query with VFILM takes on average only 1.2% more CPU, less than one standard deviation. We conclude that there is no statistically significant overhead (latency or CPU) imposed by VFILM on query operations.
P5	Cost of VDF flexibility and power (VDF execution time)	VDFCompareTest. Time to execute VDF with mission association, age, and size factors versus simple value function using only age.	4.5.10	25% increase in mean evaluation time for using 3 factors in valuation versus one factor. Nearly 3x increase in standard deviation.

4.5.2 Functional Metric F1 – ILM Responsiveness to Events

The ILM can be triggered to perform VDF valuation by Phoenix events. This test is a confirmation that Phoenix events, such as *mission start*, trigger VFILM to re-evaluate IOs associated with the mission (metric F1). The `MissionTriggersVDFTest` JUnit test confirms this by publishing 1000 IOs, split evenly between two missions. The test then sends a *mission start* event for one of the missions and confirms that exactly 500 IOs are re-evaluated by the Value Depreciation Function (VDF). It also sends a *mission start* event that does not match any published IOs and verifies that no evaluations occur in response to the event.

The HSM can be triggered to move information by Phoenix events. This test is a confirmation that Phoenix events, such as *mission prep*, can be used to trigger movement events in the HSM (specifically the ILM-HSM Adapter) (metric F1). The `MoveScalingTest` JUnit test confirms this functionality using a specialized version of the mission domain model. When the system receives a *mission prep* event, it moves a pre-defined amount of data from the level 0 store to the level 1 store. The test outputs the amount of data moved during each operation, allowing us to confirm that each *mission prep* is triggering the appropriate action.

The HSM can be triggered to move information by system events. This test is a confirmation that system events can be used to trigger movement events in the HSM (specifically the ILM-HSM Adapter) (metric F1). The only system event currently defined is a disk space event, which notifies the HSM that the level 0 store has dropped below the minimum free space threshold. The `FreespaceTriggersMoveTest` JUnit test confirms this functionality by using a 125 MB partition to store the level 0 repository and quickly publishing IOs to exhaust free space. The test logs the amount of free space left when a move event is triggered and confirms that it is below the specified threshold.

4.5.3 Functional Metric F2 – Maintaining Level 0 Store

The ILM can maintain a specific amount of free space in level 0 store. This test is a confirmation that IO movement is governed by the maximum and minimum free space thresholds specified for the system (metric F2). It is confirmed by the `FreespaceTriggersMoveTest` JUnit test. Because the test reports the amount of space freed by each move operation, it confirms that VFILM is in fact maintaining free space between the maximum and minimum thresholds defined.

Figure 41 shows a visualization of this test running. Each decrease in available space is the result of 10 MB of information being published. Along the bottom of the grid area, the total amount of information published up to that point is noted. All increases in available space are due to the ILM-HSM Adapter moving IOs from level 0 to level 1. As expected, the ILM-HSM Adapter does not move any IOs until the available space drops below the “Begin Move” threshold and it never moves enough IOs to make the free space exceed the “Stop Move” threshold.

The ILM can monitor the amount of free space in level 0 store. This hypothesis is also confirmed by the `FreespaceTriggersMoveTest` JUnit test used for the two previous tests.

4.5.4 Functional Metric F3 – Publication Correctness

A publication operation with the archive bit set will succeed in VFILM if it succeeds in the baseline Phoenix system. This test is a confirmation that all well-formed IOs will be archived successfully when published (metric F3). The `PublishPerformanceTest` JUnit test confirms this functionality by publishing a large number of IOs and then outputting the total number archived. The test was run with 1000 small IOs and 100 large IOs, all of which were archived by the repository service. The level 0 store was configured to be large enough that no move operations to level 1 were necessary during the experiment. The test confirmed that all the IOs were archived successfully in both the baseline Phoenix and in the VFILM system.

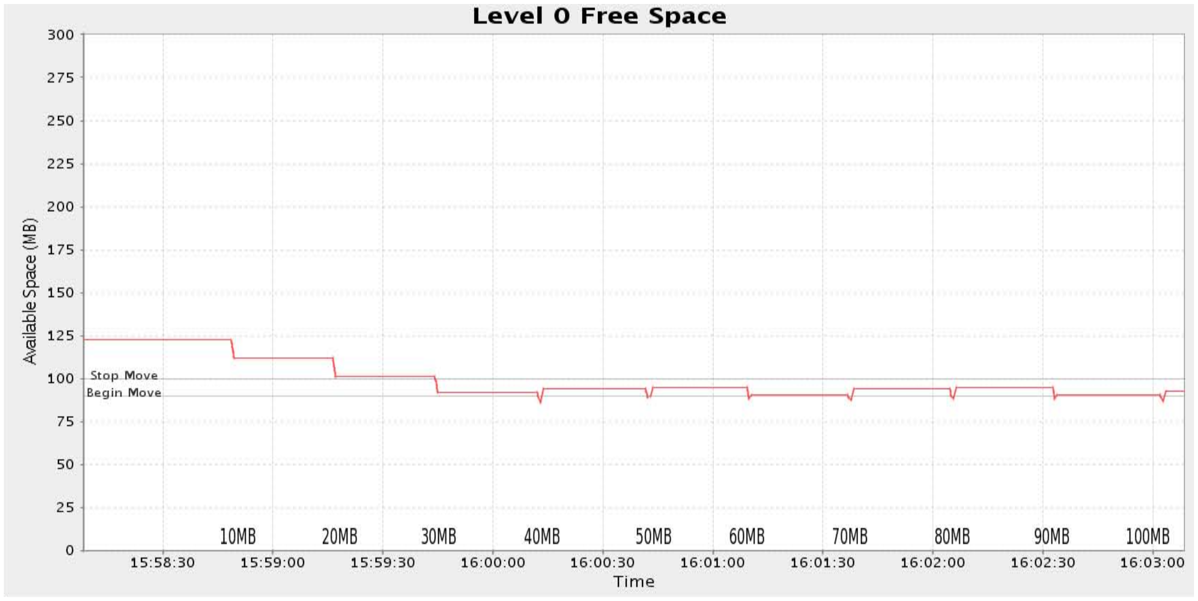


Figure 41. VFILM maintaining level 0 store: IO moves based "begin move" and "stop move" thresholds.

4.5.5 Functional Metric F4 – Query Correctness

A query operation will return the same set of results in VFILM that it does in the baseline Phoenix system. This test is a confirmation that queries will return all matching IOs in the VFILM system (metric F4). The `QueryPerformanceTest` JUnit test confirms this functionality by publishing a large number of IOs and then executing a query that returns all IOs matching a given predicate. The test was run with 1000 small IOs and 100 large IOs, all of which were returned by a query matching their predicate. The level 0 store was configured to be large enough that no move operations to level 1 were necessary during the experiment. The baseline Phoenix and VFILM systems produced the same result set.

4.5.6 Performance Metric P1 – VDF Scalability

The time to execute the VDF increases linearly with the number of objects in the evaluation set. This hypothesis covers metric P1. To test the VDF's scaling, and to confirm that it scales with the number of IOs rather than their size, the `VDFScalingTest` JUnit test publishes a predefined number of IOs and then triggers evaluation over the entire set. Using a shell script to automate the test, we ran it with two different IO sizes. For each IO size, we tested five different repository configurations, containing 1MB, 25MB, 50MB, 75MB, and 100MB total. Results for each IO size are plotted in Figure 42 and Figure 43, with a linear best fit line for each. The fit line's coefficient of determination (R^2) is also listed. The coefficient of determination is a measure of the proportion of variability in a data set that is accounted for by the fit line, with 1 being the best possible. The high variance in Figure 4 is probably due to the small number of IOs evaluated.

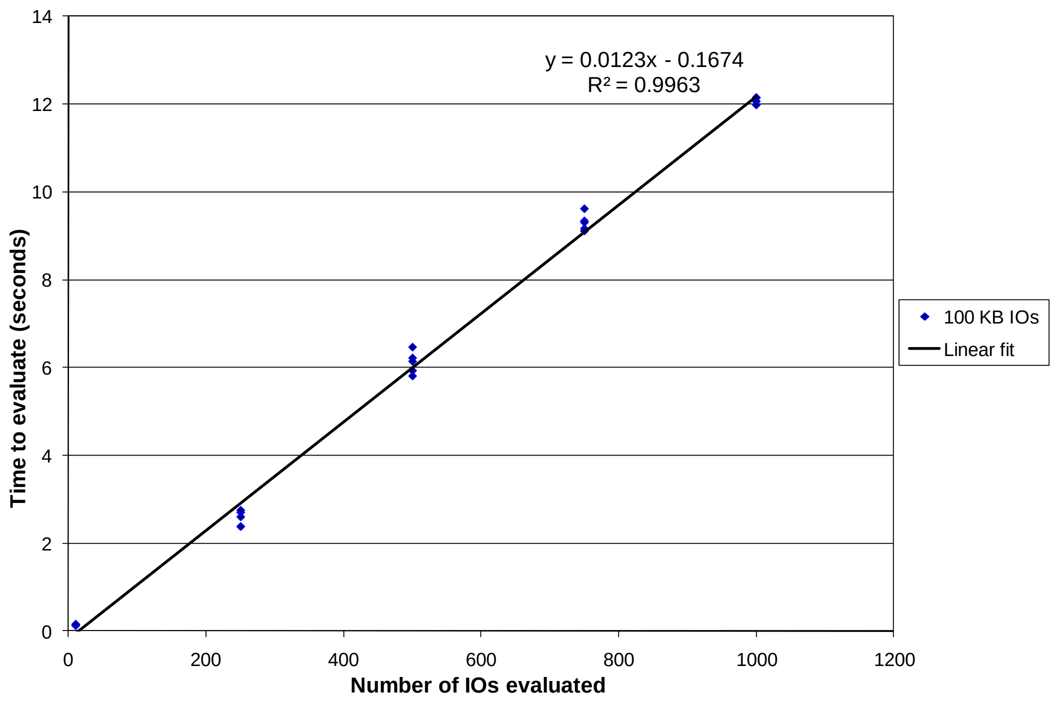


Figure 42. The time to evaluate IOs scales linearly with the number of IOs evaluated.

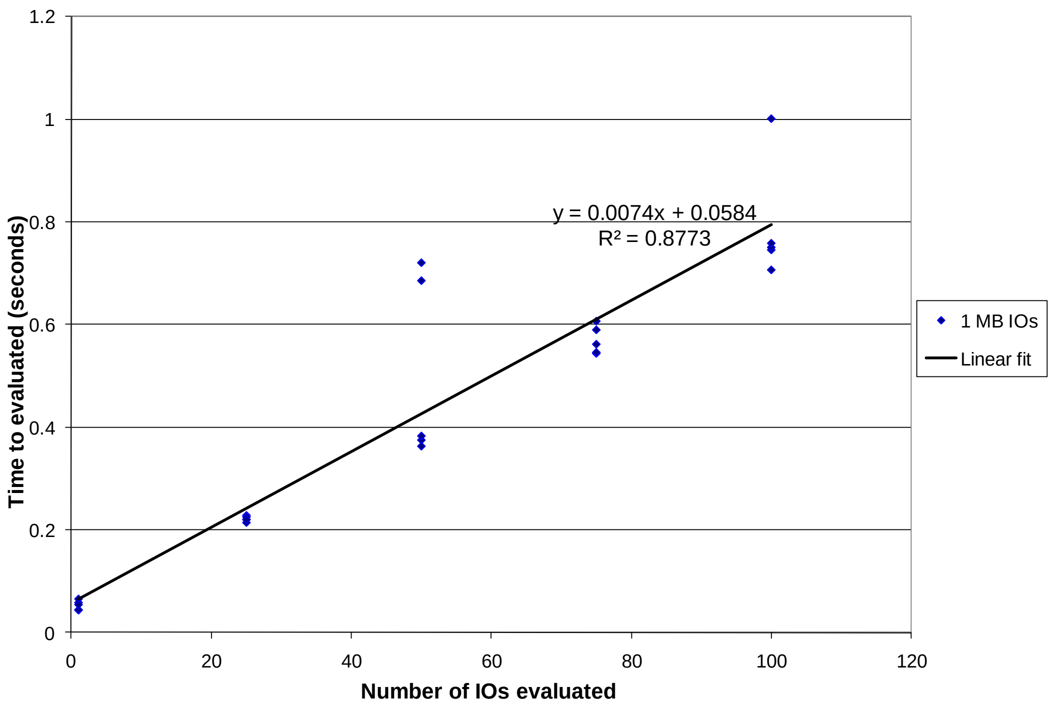


Figure 43. The time to evaluate IOs scales linearly with larger IO sizes as well.

For the smaller IO size (100 KB), there is a close linear fit. Additionally, the variance is low for all set sizes. For the larger IO size (1 MB), the scaling still appears to be linear, but the variance is much higher.

To ensure that the VDF does not scale with the size (in MB) of the evaluation set, we also plotted the results of `VDFScalingTest` in terms of the size of the set. The results for both IO sizes appear in Figure 44. The linearity of each dataset is unchanged, as the results come from the same experiment, but plotting the two series together illustrates that scaling is dependent upon the number of IOs rather than the data size of the evaluation set. With equivalent data sizes, large IOs are evaluated much more quickly because the VDF's evaluation function is called fewer times. While both the large IOs and small IOs scale linearly in this figure, they do so according to clearly different linear functions.

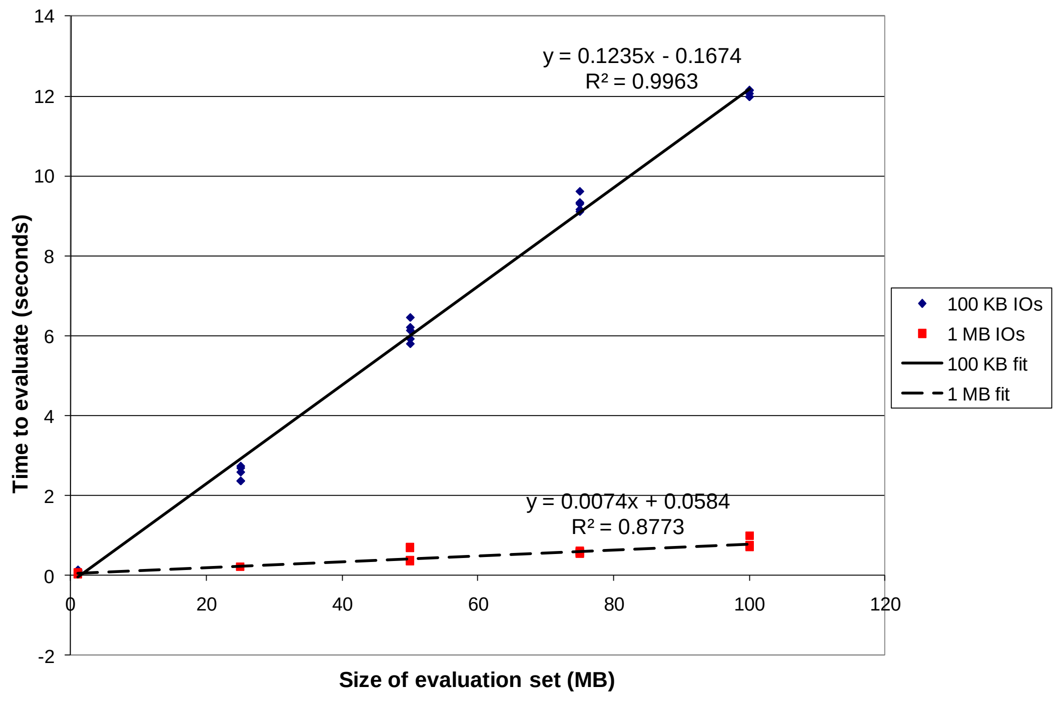


Figure 44. The VDF does not scale linearly with the size (in MB) of the evaluation set.

One observation is the difference in slope on the small (100 KB) IO VDF evaluation graph and the slope in the large (1 MB) IO VDF evaluation graph, implying that it takes longer to execute the VDF function *per IO* when the IOs are smaller. The numbers in Table 15 verify this observation.

Table 15. Comparison of the ms/IO time to execute the VDF

No. of runs	No. IOs / run	Size of IO	Ave. ms/IO	Overall Ave. ms/IO
5	25	1 MB	8.856	8.623833
5	50	1 MB	10.112	
5	75	1 MB	7.5973	
5	100	1 MB	7.93	
5	250	100 KB	10.2144	11.72483
5	500	100 KB	12.214	
5	750	100 KB	12.40693	
5	1000	100 KB	12.064	

The average time to execute the VDF per IO appears to increase with the number of IOs in the evaluation set. The sample size of these runs is too small (in terms of the variations in numbers of IOs) to draw too many conclusions, but there is an appearance of an increase in the average execution time of the VDF as the evaluation set increases, with an apparent 35% or so increase in execution time when moving from 10s of IOs to 100s of IOs. This could be due to the cost of storing and retrieving larger amounts of index information (e.g., paging effects), but we do not have enough information to diagnose or draw too many conclusions. When the opportunity comes to transition VFILM into experimental or operational environments, additional experiments and analysis is warranted to determine how well the VDF scales to much larger sets of IOs (e.g., thousands or millions).

The conclusion might point to places for optimization of the VDF function (we have suggestions for potential optimizations in Section 4.6) or to usage patterns, such as running VDF valuation over groups of IOs instead of over full storage systems.

4.5.7 Performance Metric P2 – HSM Scalability

The time to execute the HSM move operation increases linearly with the number of objects moved. This hypothesis covers metric P2 and it is only true when all IOs are of the same size. We tested it using the `MoveScalingTest` JUnit test, which was run with two different IO sizes and five different repository sizes, as specified in the experiment plan. The results for both IO sizes appear in Figure 45. The coefficient of determination indicates a strong linear correlation in both cases. Figure 45 also clearly shows that the two IO sizes do not scale according to the same linear function. It takes longer to move an equivalent number of large IOs because the HSM executes the same number of move operations, but each operation requires more data to be moved on the file system. While both large and small IO sizes appear to scale linearly, they do so according to different linear functions.

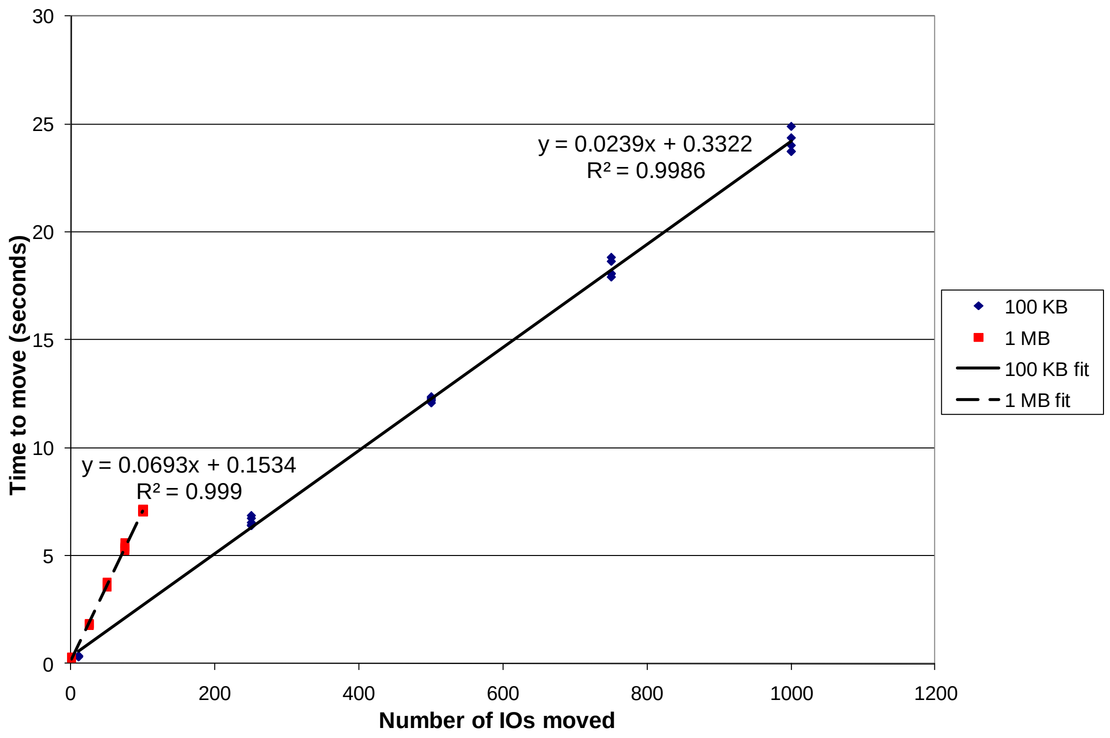


Figure 45. The HSM only scales linearly when all IOs are of the same size.

The time to execute the HSM move operation increases linearly with the number of bytes moved. This hypothesis covers metric P2. The test refutes this hypothesis with the results re-plotted in terms of data size in Figure 46. We only observed linear scaling when all IOs are of the same size, which is not likely to be the common case. Large IOs are moved more quickly because the HSM needs to execute fewer move operations in order to free the same amount of space. The overhead of locating files on disk to move dominates the run time. The size of each IO also affects the scale, as this plot shows. It requires fewer move operations on the file system to free the same amount of space with large IOs.

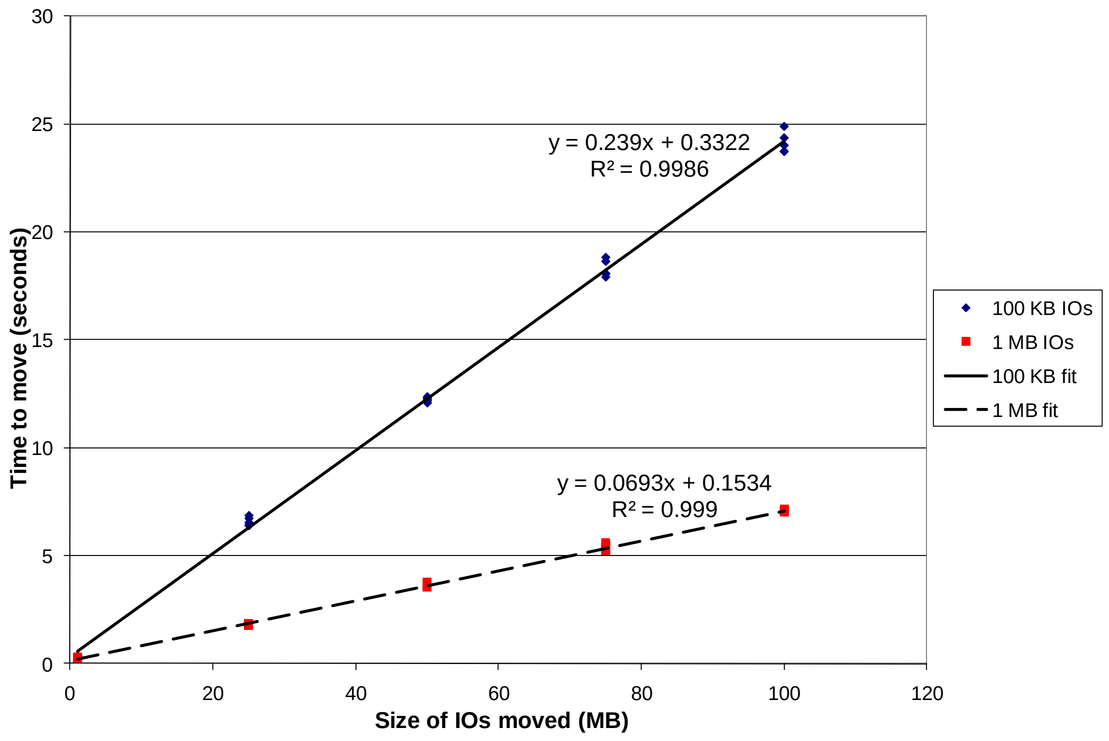


Figure 46. The HSM does not scale linearly with the number of bytes moved.

4.5.8 Performance Metric P3 – Publication Performance

A publication operation will take approximately the same amount of time to execute in VFILM as it does in the baseline Phoenix system. The `PublishPerformanceTest` JUnit test confirms this hypothesis. It publishes 1000 IOs and runs a timer starting at the first publication operation and stopping at the last repository insertion. Table 16 shows the time reported by this internal timer, as well as the real time and CPU usage according to the UNIX time utility. CPU usage greater than 100% is possible because the test server has multiple processors.

Table 17 shows a summary of some relevant results from this experiment. The mean insertion time between Phoenix and VFILM differs by about 1.5 standard deviations (taking the Phoenix standard deviation), which indicates that it is plausible for the run times to have come from the same statistical population. The CPU usage, however, is clearly different in a statistically significant degree. Again taking the standard deviation from Phoenix, the CPU usage means are nearly ***six*** standard deviations apart. This is attributable to the additional overhead of running VFILM's Spiral 1 prototype, which had three additional services running than the baseline (the ILM service; a separate Group Manager service, which was folded into the ILM service in Spiral 2; and the Phoenix Event Notification Service, which was not running in the baseline).

There are a few other issues worth noting in these results. The standard deviations are lower for VFILM than for Phoenix, which is probably due to testing on a multi-user system. Also, the "real time" numbers recorded are much larger than the "time to insert" because the real time measurements include short delays between publication operations.

Table 16. Phoenix publication performance and VFILM publication performance.

Phoenix Publication of 1000 IOs				VFILM Publication of 1000 IOs		
Time to insert (secs)	Real time (secs)	CPU usage (%)		Time to insert (secs)	Real time (secs)	CPU usage (%)
52.67	78.3	108		48.22	75.5	125
51.62	77.4	110		47.28	74.9	126
48.59	74.2	114		48.32	76.2	125
49.08	74.5	113		47.89	75.3	126
50.41	75.88	112		47.87	75.3	127

Table 17. Summary of Phoenix vs VFILM publication performance.

		Phoenix	VFILM
Time to insert (seconds)	Mean	50.47	47.92
	Standard deviation	1.705	0.4087
CPU usage (%)	Mean	111.4	125.8
	Standard deviation	2.408	0.8366

4.5.9 Performance Metric P4 – Query Performance

A query operation will take approximately the same amount of time to execute in VFILM as it does in the baseline Phoenix system. The `QueryPerformanceTest` JUnit test confirms this hypothesis. It publishes 1000 IOs, waits for all IOs to be inserted into the repository, and then issues a query that returns the entire set. A timer is started when the query is executed and stopped when the last IO is returned from the repository. Table 18 shows the time reported by this internal timer, as well as the real time and CPU usage according to the UNIX time utility. CPU usage greater than 100% is possible because the test server has multiple processors.

Table 19 shows a summary of some relevant results from this experiment. The mean query time between Phoenix and VFILM varies by less than 1.5 standard deviations (taking the Phoenix standard deviation), which indicates that it is plausible for the run times to have come from the same statistical population. The CPU usage is within a single standard deviation. Query performance is much closer than publication performance for the two systems because the extra services running in the VFILM experimental case do not do any processing in response to queries.

Table 18. Phoenix query performance and VFILM query performance.

Phoenix Query for 1000 IOs				**VFILM Query for 1000 IOs**		
Time to re-trieve (secs)	Real time (secs)	CPU usage (%)		Time to re-trieve (secs)	Real time (secs)	CPU usage (%)
13.98	39.42	161		14.12	41.18	160
13.88	39.24	160		14.11	40.82	164
13.67	38.86	164		14.04	40.67	166
13.88	39.33	164		13.88	40.55	162
13.68	39.1	162		13.8	40.60	165

Table 19. Summary of Phoenix vs. VFILM query performance.

		Phoenix	VFILM
Time to process query (seconds)	Mean	13.82	13.99
	Standard deviation	0.1369	0.1452
CPU usage (%)	Mean	162.2	163.4
	Standard deviation	1.789	2.408

4.5.10 Performance Metric P5 – Mission Effectiveness

Using a combination of factors to determine which IOs have the highest value will result in more "mission-relevant" IOs in level 0 store without unduly affecting performance. VFILM's use of FCL offers flexibility advantages over a simpler valuation scheme. Valuation rules, written in FCL, calculate the value of an IO using its group associations and properties of the IO. For example, the current VDF is based upon mission association, age, and size. Additionally, the rules can be changed at runtime to reflect changing deployment requirements. For instance, any factor can be dropped or have its weight adjusted.

The `VDFCompareTest` JUnit test allows us to estimate the cost of the VDF flexibility described in the previous paragraph versus a simpler, single-factor valuation function. The simple function that we used for experiments ranked IOs based on age. We used `VDFCompareTest` to publish and evaluate 1000 IOs, reporting the total time spent evaluating all IOs. Table 20 shows the mean and standard deviation across ten runs. Based on the mean values, the VDF increases the evaluation time by about 25% compared to the single-factor function.

Table 20. The time (in milliseconds) spent in the evaluation function for both the single-factor value function and the fuzzy logic VDF.

	Single-factor function	Fuzzy logic VDF
Mean evaluation time	15.98 s	19.97 s
Std. deviation	84.27 ms	249.1 ms

4.5.11 Summary of Experimental Results

This section has presented the results of experiments to evaluate the functionality and performance of the current VFILM prototype software. The process of defining the metrics, constructing and executing the tests, and analyzing the results has the following significant conclusions:

- We have produced a set of JUnit tests used to conduct the experiments herein, which serve as a set of validation and performance tests useful as part of the software and documentation delivery for this prototype.
- Many of the testing results, specifically the functional tests and metrics, validate that the VFILM prototype meets the functionality requirements that we set out to implement.
- Other testing results, specifically the scalability metrics, indicate more clearly the factors that go into the performance of the ILM and HSM components of VFILM, and establish a prototype baseline for comparison with later versions or prototypes, e.g., that interact with different repositories, file systems, or HSM systems.
- The performance results indicate that the Spiral 1 VFILM prototype, developed as a rapid prototype and not optimized for performance, actually imposes very little overhead on the Phoenix publication, archive, and query operations.

The performance test comparing multi-valued VDF evaluation using fuzzy logic versus single valued evaluation provides two significant conclusions. First, since the prototype was developed with flexibility, extensibility, and expressiveness in mind, not performance, the additional time we observed does not appear to be too bad. Second, however, when contrasted with the other performance results, it indicates an opportunity for optimization in future versions.

4.6 Lessons Learned & Recommendations

One of the goals of the VFILM prototype was to provide a general Information Lifecycle Management framework. Because of this, we tried to make few assumptions about use case scenarios and deployment situations. This, however, resulted in us making very few optimizations to the prototype software. In this section, we discuss several possible optimizations that we can recommend.

Not retrieving entire Information Objects for evaluation. When an IO needs to be reevaluated the current prototype retrieves the entire IO from the disk and passes it to the VDF. However, the evaluation rules only use certain characteristics: an IO’s mission status, creation time, and size in the VFILM demonstration. Mission status (and other group memberships) was based on a specific set of fields in the ILM Index (IO Type, publisher ID, location, etc.). Group status could have been determined based upon these indexed fields without reading the entire IO from the disk every time it had to be evaluated. Similarly, IO creation time and IO size could be stored in the Index (or another suitable DB) and passed to the VDF without reading the entire IO from the disk. For IOs with particularly large payloads this could drastically increase performance.

Using the Repository as the ILM Index. The ILM Index and Accessor Interface were designed so that information groups could be defined over any possible fields. To handle this, we introduced an additional database to keep track of IOs. In some scenarios it could be possible that all relevant grouping information is contained in an IO's metadata. In these situations, the repository's metadata database could provide the functionality of the ILM Index. Because the ILM Index is contained in the ILM-HSM Adapter, this would simply require modifying the adapter to make group related queries against the repository's metadata DB instead of the ILM Index. This would reduce the overhead of the ILM Service by removing the overhead associated with the ILM Index.

Using the Repository as the Value Store. The BerkeleyDB Value Store utilizes a Berkeley Database for storing the ID, value, and storage location of an IO. This is information that could be potentially stored in the Repository's metadata database. Because the value store is managed by the ILM-HSM Adapter, calls to the value store could be redirected to the Repository instead and the interface to the Adapter would remain the same.

Bulk operations. There are several points in the VFILM prototype where IOs are treated in batches (evaluation, indexing, moving, retrieval from repository, etc.). Values for the size of the batches were generally picked arbitrarily and we did not tune the configuration for performance. The following list is a few places where such values could be tuned for better performance:

- Index – When the group status of IOs is being retrieved, this could be done in batches of *n* IOs at a time.
- VDF – When the valuation of IOs is being performed, this could be done over batches of *n*.
- BerkeleyRepoAdapter – During moves and clean up operations, IO entries from the value store could be retrieved in batches of *n*.

Information Channel for Retrieved IOs. Currently when the ILM-HSM adapter requests IOs from a repository, these are returned via a call to the ILM compatible Repository Service. Although this does what is required, it does not match Phoenix's use of Information Channels for passing IOs between services, and is an area for enhancement.

4.7 Directions for the Future

While this project produced significant results and established a solid prototype ILM service, it has only scratched the surface of the potential for ILM within the AFRL suite of Information Management capabilities. This section describes some potential future directions in which research, development, and transition activities could proceed.

Transition and Experimentation. The experiments that we conducted and the metrics that we gathered provide some concrete and empirical evidence of the VFILM functionality and performance, as well as some areas for emphasis in looking to optimize it. A next step would be to try it out in closer-to-real environments, e.g., embedded platforms like Marti, the LITENING Pod [14], or one of the Navy Limited Technology Experiments (LTE).

Another path forward is to integrate the VFILM software into the Phoenix code repository, and make it a mainline part of the Phoenix software distributions. Some of the work we did in the VFILM effort, such as the effort to make VFILM work with the existing Phoenix services and the provision of JUnit tests for VFILM, help facilitate this path forward.

Support for Additional Classes of Repositories. In the prototype that we developed, we took advantage of some of the ways that the Berkeley Repository is implemented and the ways in which it is used in the Phoenix services. However, the Berkeley Repository is just one possible Phoenix repository, and different repositories will have different requirements and potential optimizations. For example, whereas the Berkeley repository stores one IO per file and supports any Information type, the PostGIS repository used in Fawkes stores all IOs in a database table and manages only one Information type, i.e., Cursor on Target [24].

Improvements to the Phoenix Query Service to Support Repositories that Might be Located in Multiple Hierarchical Levels of Storage. Whereas we made VFILM as non-intrusive to the existing Phoenix services as possible, the introduction of ILM has visible side effects on the Phoenix Query Service, in the form of the following:

- Potentially returning a different order of results when IOs are moved
- Introducing latency when queries involve non-level 0 storage
- Potentially a reduced result set when IOs are moved to non-indexable storage

These effects happen and can be visible to the user, but without him/her having any control over them, unless the Query Service is extended to be ILM-aware and to expose interfaces to manage the options that ILM provides.

One approach is that the Query Service could respect "hints" provided by the querying client. The hints could be stored in the ILM Repository context or the ILM Query context and could include options such as the following:

- The Query Service should provide results in level 0 first.
- The Query Service should provide only results in level 0.
- The Query Service should return all results.
- The Query Service should execute queries over specific storage levels.

Another useful feature would be a Query Service interface that could provide statistics such as the following that would be useful to the querying client:

- The number of levels of storage.
- The number and types of IOs at each level.

Expansion of Mission Models and VDF Factors. The VFILM prototype includes straightforward versions of some of the concepts that it covers. For example, the mission model that we cover includes essentially only the following mission aspects:

- The identification of mission epochs, specifically preparation for an upcoming mission, mission start, and mission end.
- The realization that some missions are more important than others (captured in the importance in VFILM policy).

There is the opportunity in the future to examine much more comprehensive mission models. We can only anticipate what some of these models might include, and certainly the focus should be on the aspects of the models that affect information lifecycle decisions. Some of the aspects that we could envision being captured in such mission models include, but are not limited to, the following:

- The types, characteristics, and rates of information used in the mission, which can affect the depreciation factors and values used in the valuation function. For example, a mission involving rapidly unfolding situations (e.g., Time Critical Targeting) might include high rates of information, with new information rapidly supplanting old information, and aggressive devaluation and movement of information.

- Groups of information associated with the mission and/or rules governing the formation of groups.
- The relative importance of operations within the mission, e.g., a mission might include real-time data streams being processed by the IM system through publication and subscription (the Submission and Brokering services), storage and retrieval of archived information (the Repository and Query services), and Command/Control traffic (Event processing). Particular missions or parts of missions can rely more heavily on one or more of these operations, and the ILM system can use this knowledge in scheduling valuation and movement, and in setting the storage thresholds.

Use of VFILM Fuzzy Decision Algorithms in Other Contexts. In this project, we made a good case for the use of fuzzy logic based functions for decision making of particular types, specifically those which need smooth transitions based on partial orders and with relative valued inputs. These same reasons are why fuzzy logic is used in subway, thermostat, and elevator algorithms. There are IM software adaptive decision engines that might benefit from fuzzy logic based algorithms, also, including two areas in which AFRL is funding research: QoS management and adaptive security.

Coordinating Valuation Function between Multiple ILM Instances. When IM services are distributed and federated, they can include multiple and distributed ILM instances. Additional semantics and configuration options can result in more effective ILM services and, subsequently, more effective IM services. While this is a rich area of research, a few initial thoughts follow.

Consider a deployment in which an IM system (or multiple instances of an IM system) utilizes multiple repositories but with a single ILM service running one instance of the VDF, as shown in Figure 47. If information is duplicated in the repositories, then the single VDF function will distribute the information in both repositories approximately the same way.

However, if the ILMs use different, but coordinated, VDFs, IOs can be distributed within the repositories so that each contains a different set of IOs in level 0 storage, effectively increasing the number of IOs available, as shown in Figure 48.

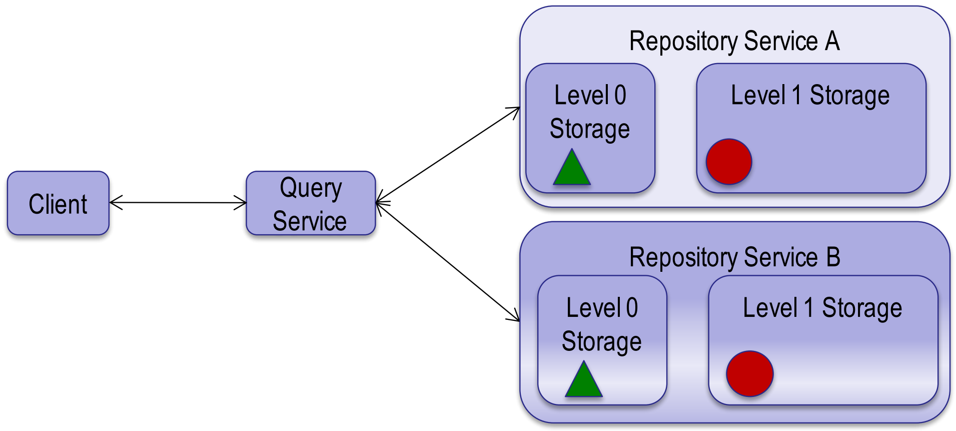


Figure 47. Using the same valuation function between multiple ILM instances will result in approximately the same IO distribution.

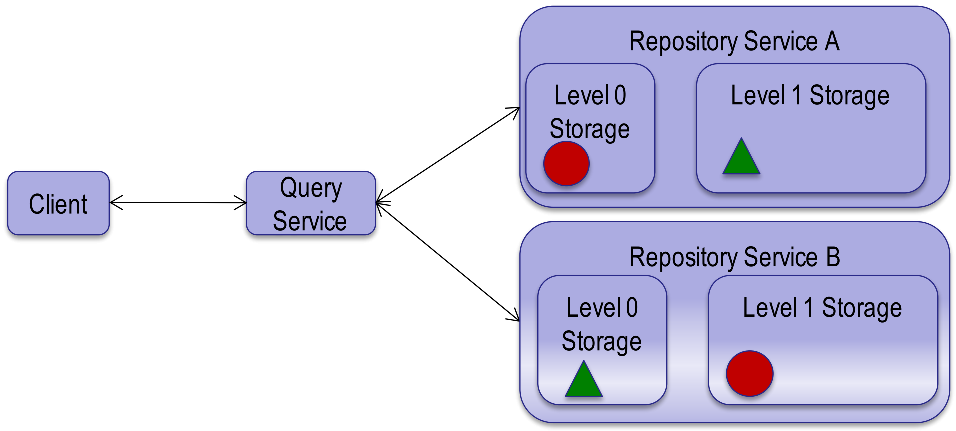


Figure 48. Using different valuation functions can result in a better distribution of IOs in level 0 store across multiple repositories.

Knowledge of resource availability and QoS awareness can be combined with the ILM capabilities to base the IO distribution on the needs of clients. As shown in Figure 49, if there are multiple clients using an IM system (or distributed IM services) with multiple repositories across multiple storage levels, correctly crafted and coordinated VDFs can ensure the IOs most critical to each client are distributed into the repository most accessible to the client.

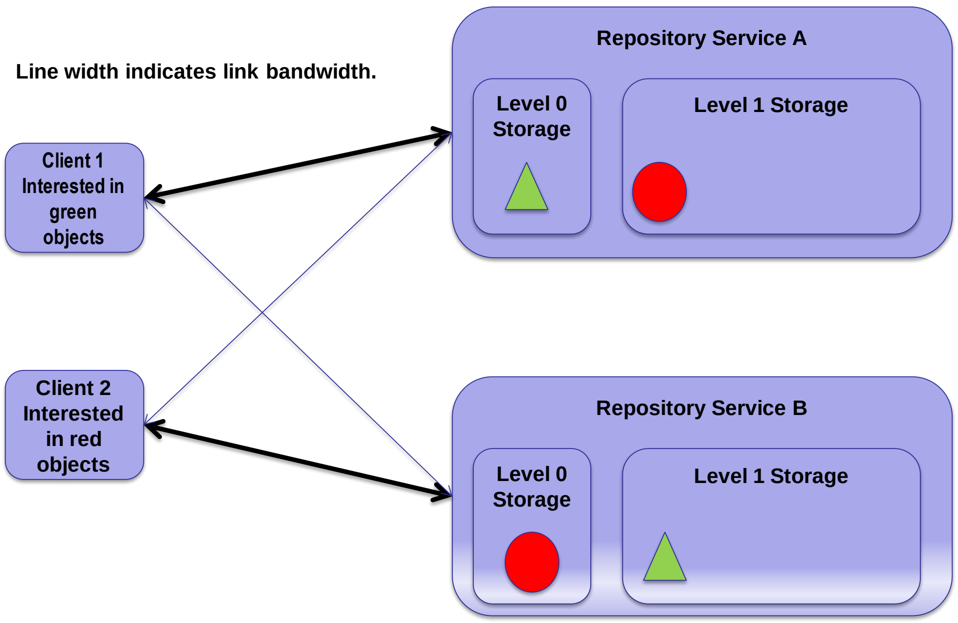

Line width indicates link bandwidth.
Repository Service A
Level 0 Storage
Level 1 Storage
Client 1 Interested in green objects
Client 2 Interested in red objects
Repository Service B
Level 0 Storage
Level 1 Storage

Figure 49. Network topography, endpoint requirements, and bandwidth availability can be taken into account to move IOs close to the clients that need them.

5.0 CONCLUSIONS

The VFILM project was a successful research effort that produced significant advancements in the investigation, prototyping, and evaluation of mission-driven information lifecycle management for IM services. The architecture, design, and prototype software that we developed under this project provide a foundation for ILM in enterprise and tactical environments.

The VFILM architecture and design include triggering of information lifecycle management based on mission events and mission-based policy, valuation of information using fuzzy logic algorithms based on the information's urgency to ongoing mission operations, grouping of information based on common attributes and dependencies, and migration and retrieval of information objects and groups.

The major contributions of the VFILM project include:

- A prototype ILM service that provides mission-aware information valuation, control of HSM movement of information between levels of storage, and support for AFRL Phoenix IM services, Information Objects, and repositories.
- An ILM-HSM interface that abstracts the details of specific HSMs, file systems, and repositories.
- A novel approach to information valuation, supporting an extensible multi-factor assessment of the relative values of information using fuzzy logic. The approach produces a partial order of information depreciation, handles dynamic conditions that can change the worth of information, and avoids the thrashing that is possible with fixed or static valuation thresholds.
- A set of experimentation results and unit tests, which are useful as a functional and performance test suite for ILM services.

While the focus of this project was on the research and rapid prototyping of ILM capabilities, the prototype software that we developed is a useful result, with sufficient functionality to explore full integration with Phoenix capabilities and transition into experimental testbeds or demonstration platforms.

Further research building upon this foundation can explore additional richness in the VFILM prototype, e.g., to expand its mission models and the factors utilized in valuation; expanding the query service to be more aware of the hierarchical storage levels and to exploit this knowledge to order query responses; and to explore distributing and coordinating ILMs for improved storage and access to critical information.

6.0 REFERENCES

[1] Constantin von Altrock, “Fuzzy Logic and NeuroFuzzy Technologies in Appliances,” *Embedded Systems Conference*, 1996.

[2] Ying Chen. Information valuation for information lifecycle management. 2nd Intl. Conf. On Autonomic Computing, June 13-16, 2005.

[3] Jeffrey Cleveland, Shane Clark, Joseph Loyall, Jonathan Webb, “Value Factor based Information Lifecycle Management Installation, Operations, Administration, & Demonstration Guide,” October 28, 2010.

[4] Christine Taylor Chudnow. Information lifecycle management and the government. *Computer Technology Review*, August 1, 2004.

[5] Cluster File Systems, Inc. “Lustre: A Scalable, High-Performance File System.” http://eugen.leitl.org/whitepaper.pdf.

[6] http://dvdvault.sourceforge.net/

[7] -, “Fuzzy Logic,” *Stanford Encyclopedia of Philosophy*, August 4, 2010, http://plato.stanford.edu/entries/logic-fuzzy/.

[8] Jörg Gebhardt, Constantin von Altrock, “Recent Successful Fuzzy Logic Applications in Industrial Automation,” *Fifth IEEE International Conference on Fuzzy Systems*, September 1996, New Orleans, LA.

[9] Robert Grant, Vaughn Combs, James Hanna, Brian Lipa, James Reilly, "Phoenix: SOA Based Information Management Services," Proceedings of the 2009 SPIE Defense Transformation and Net-Centric Systems Conference, Orlando, Fl, April 2009.

[10] Ming Ho, B. Robertson, “Elevator Group Supervisory Control Using Fuzzy Logic,” Canadian Conference on Electrical and Computer Engineering, September 25-28, 1994, Halifax, NS, Canada.

[11] International Electrotechnical Commission (IEC), “IEC 1131 – Programmable Controllers, Part 7 – Fuzzy Control Programming,” January 1997.

[12] Bart Kosko, Satoru Isaka, “Fuzzy Logic,” *Scientific American*, July 1993.

[13] Richard Kugler, Michael Baranick, and Hans Binnendijk. Operation Anaconda lessons for joint operations. Center for Technology and National Security Policy, National Defense University, March 2009.

[14] -, LITENING, Advanced Airborne Targeting and Navigation Pod. Federation of American Scientists Military Analysis Network. October 28, 1999, http://www.fas.org/man/dod-101/sys/smart/litening.htm.

[15] Joseph Loyall, Jeffrey Cleveland, Jonathan Webb, Shane Clark, “Value Factor based Information Lifecycle Management Experiment Plan,” August 11, 2010.

[16] Joseph Loyall, Shane Clark, Jeffrey Cleveland, Jonathan Webb, “Value Factor based Information Lifecycle Management Experiment Results,” August 11, 2010.

[17] Joseph Loyall, Matthew Gillen, Aaron Paulos, Larry Bunch, Marco Carvalho, James Edmondson, Douglas Schmidt, Andrew Martignoni III, Asher Sinclair, “Dynamic Policy-Driven Quality of Service in Service-Oriented Information Management Systems,” *Software: Practice and Experience*, 2011.

[18] http://wiki.lustre.org/index.php/Main_Page

[19] -, “MASSIVE 4.0, Supporting V-Ray Rendering, is Now Available,” *Business Wire*, February 18, 2010, http://www.thefreelibrary.com/MASSIVE+4.0,+Supporting+V-Ray+Rendering,+is+Now+Available-a0219126342.

[20] Mitsubishi Electric, AI Supervisory Control System, http://www.mitsubishi-elevator.com/innovations/control_system.html.

[21] http://openhsm.sourceforge.net

[22] http://www.opensolaris.org/os/project/samqfs/

[23] http://opensolaris.org/os/project/adm/WhatisADM/

[24] D. Robbins. Unmanned Aircraft Operational Integration Using MITRE's Cursor on Target. The Edge, MITRE, Volume 10, Number 2, Summer 2007.
[25] Beth Pariseau. Does EMC's ILM strategy really solve complexity? *Storage Technology News*, August 30, 2005.
[26] Thomas E. Sowell, "Fuzzy Logic for "Just Plain Folks"", http://www.fuzzy-logic.com/.
[27] Jessika Toothman, "How Rice Cookers Work," http://home.howstuffworks.com/rice-cooker2.htm.
[28] L.A. Zadeh, "Fuzzy Sets," *Information and Control*, 8, 338-353 (1965).

7.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AFRL	Air Force Research Laboratory
ADM	Automatic Data Migration
BFT	Blue Force Track
CSV	Comma Separated Value
DB	Database
FCL	Fuzzy Control Language
FFLL	Free Fuzzy Logic Library
FIS	Fuzzy Inference System
GB	Gigabyte
GUI	Graphical User Interface
HSM	Hierarchical Storage Management
ID	Identifier
IEC	International Electrotechnical Commision
ILM	Information Lifecycle Management
IM	Information Management
IMS	Information Management Services
IO	Information Object
ISQM	Information Space QoS Manager
ISR	Intelligence, Surveillance, and Reconnaissance
MB	Megabyte
MDDB	Metadata Database
MPEG-2	Moving Picture Experts Group Standard Video Encoding Format Version 2
NAS	Network Attached Storage
PDD	ProData
PI	Principal Investigator
PM	Program Manager
QED	QoS Enabled Dissemination, another AFRL project led by BBN Technologies
QoS	Quality of Service
RAM	Random Access Memory
RMAN	Recovery Manager
SAN	Storage Area Network
SATA	Serial Advanced Technology Attachment
TIM	Technical Interchange Meeting
VDF	Value Depreciation Function
VFILM	Value Factor driven Information Lifecycle Management
XML	Extensible Markup Language

APPENDIX – CHRONOLOGICAL ACCOUNT OF TECHNICAL STATUS

This section describes the chronological technical status as reported monthly during the Enterprise Information Lifecycle Management contract. These are illustrative of the evolution of the research results, design, and implementation over the twelve month period of the contract.

Inception through November 2009

During the report period, we kicked off the VFILM project. We established the following to help facilitate research and development:

- A Wiki
- A code repository in Subversion

We scheduled the kickoff meeting with AFRL for December 8 and began preparing for it.

We established accounts on the AFRL Jiffy reporting system for providing deliverables to AFRL.

We began designing the interfaces and algorithms for the VFILM prototype Information Lifecycle Management service. We also began investigating existing hierarchical storage management systems to evaluate the common features that should be simulated and their suitability for use in VFILM. These will be presented in the kickoff meeting with AFRL.

December 2009

During the report period, we achieved the following technical accomplishments:

- We revised the ILM interface definition.
- We revised the vdf() algorithm.
- We analyzed the VFILM metrics from the VFILM proposal and made some changes that improve the metrics' evaluation of VFILM success and make them more reasonable to collect.
- We conducted some additional analysis of existing hierarchical storage management capabilities.

We held a kickoff meeting with AFRL on December 8, at which we presented the following:

- Motivation and goals of the project.
- Schedule, tasks, and deliverables.
- Our planned technical approach.
- The planned current tasks and architecture of the planned VFILM prototype.

January 2010

During the report period, we achieved the following technical accomplishments:

- Implemented an initial ILM service.
- Built Phoenix and the ILM service and got them running in Java 7. Java 7 includes the NIO.2 filesystem, which provides more control over the movement and copying of files, monitoring of directories and files, support for symbolic links, better scaling for large directories, and other features that will prove useful for implementing HSM functionality or

an interface to HSM functionality. Java 7 is also backwards compatible with the previous IO system.

- Investigated the Phoenix Query and Repository services and Berkeley XML DB usage.
- Refined the VFILM design to take advantage of the current Phoenix capabilities to support rapid prototyping.
- Investigated the Lustre file system for its suitability to support a simulated HSM.
- Designed an ILM-HSM interface and began prototyping it.
- Produced slides for the AFRL technology council briefing scheduled for February.
- Scheduled the next technical interchange meeting with AFRL for February 23, 2010.

February 2010

During the report period, we focused on development of the VFILM prototype and hosting a technical interchange meeting with AFRL to report on progress to date.

We refined the design of the ILM service to identify the following four key components, shown in Figure :

- An *ILM Event Manager* that listens for incoming mission/system events and translates them into internal ILM events.
- An *ILM Controller* that reacts to ILM events, updates the value depreciation function, triggers IO evaluations, and triggers HSM actions.
- The *Value Depreciation Function* that evaluates information objects using a specified policy.
- An *ILM-HSM Adapter* that abstracts away the specifics of the HSM and Phoenix Repositories being used.

We implemented an initial prototype ILM Event Manager that receives events through a Phoenix Event Channel. We also defined an initial set of ILM events that includes the following:

- *NeedSpace* – An event indicating that a specific amount of space in Level 0 store will be needed.
- *Cleanup* – An event indicating that a specific amount of time is available to perform information valuation and/or movement.
- *MaintainFreeSpace* – An event indicating that a specific amount of free space should be maintained in Level 0.
- *InfoValued* – An event indicating that something has happened that increases the urgency (i.e., the valuation) of some set of information.

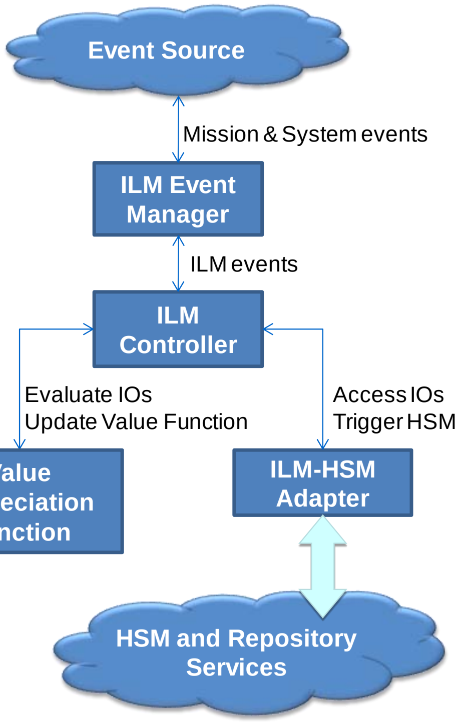


Figure A-1. Design of the ILM Service as of February 2010.

- *InfoDevalued* – An event indicating that something has happened that reduces the urgency of (i.e., devalues) a set of information.

We also identified an initial set of example mission and system events associated with operations or system conditions that are recognizable to a user or operator, and that map to the above ILM events. These include the following:

- *MissionPrep* – Preparing for an upcoming mission or operation, in which the *NeedSpace* ILM event might be triggered to free up an amount of space that that mission or operation is expected to need, or the *Cleanup* ILM event to free up as much space possible until the mission or operation commences.
- *MissionBegin* – The start of a mission or operation, which could trigger the *InfoValued* ILM event to increase the valuation of information associated with the mission or operation, or the *NeedSpace* ILM event to free up space that the mission or operation needs.
- *MissionEnd* – The end of a mission or operation, which could trigger the *InfoDevalued* ILM event for information associated with the mission or operation.
- *ThresholdCrossed* – A system event indicating that the repository size has exceeded a specific threshold, which could trigger the *MaintainFreeSpace* ILM event to move enough information to return to the desired level of free space.

We also developed an initial prototype of a Value Depreciation Function built using *jFuzzyLogic*, an open-source fuzzy logic package. jFuzzyLogic is written in Java and implements Fuzzy Control Language, standardized by the International Electrotechnical Commission in standard IEC 1131-7.

The VDF implementation consists of the following three components:

- The fuzzy sets representing the inputs (e.g., IO size, age, relevance) and output (i.e., the partial order valuation) to the VDF function.
- A set of fuzzy logic rules that combine the fuzzy inputs into a degree of membership in the output set.
- A set of Java objects for accessing the "real world" values for the fuzzy inputs, stored in information metadata, information or other Context objects, system attributes, global state (i.e., the *world context*), or other places.

In the upcoming months, we will be defining the actual input sets and fuzzy logic rules that will implement a demonstrably useful VDF function.

We refined the design of the ILM-HSM Adapter. As part of this, we defined an interface that we believe will support a variety of HSM implementation options. We plan to implement our initial prototype to work with the current Phoenix Berkeley DB implementation in the following ways (shown in Figure):

- Takes advantage of a separate Metadata Database (MDDB) and each Information Object being stored in a separate file. Both of these are true in the Phoenix 1.4.6 repository.
- Maintains a separate Level 0 store (not under HSM control) and Level 1 store (under HSM control).
- When an Information Object should be moved, the ILM-HSM adapter physically moves the IO's file from the Level 0 filesystem to the Level 1 filesystem, placing the under HSM control.

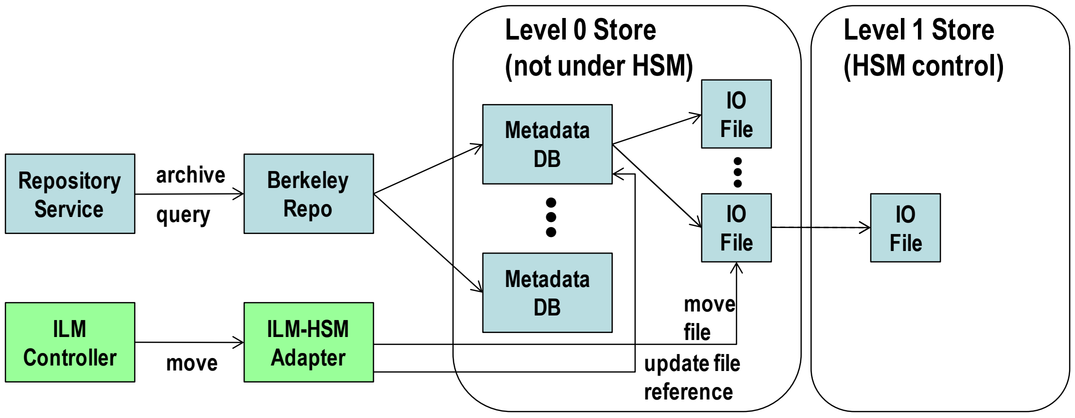


Figure A-2. Design of the ILM-HSM Adapter as of February 2010.

- The ILM-HSM adapter then updates the file reference for the IO in the MDDB to reflect the new location.

We investigated Lustre further to determine its suitability as an HSM. Lustre is a distributed file system, not an HSM solution. Lustre has an HSM project, but its focus is not the provision of an HSM, but instead to produce an interface for the Lustre filesystem to use other HSMs, similar to the ILM-HSM adapter that we are prototyping. Currently, there is no implementation available for the Lustre HSM. The Lustre HSM project does specify a policy engine, the open source Robin Hood policy engine, which could potentially be used to control archiving and retrieving IO files. Our current approach is to develop the ILM-HSM adapter and simulate the HSM functionality, to not be gated by any external HSM development effort. We also expect that Lustre is not suitable for Level 0 storage since it is a distributed filesystem, but that it is an option for higher level storage.

March 2010

During the report period we focused on development of additional components of the VFILM Information Lifecycle Management service, shown in Figure .

During the report period, we developed initial prototypes of the ILM-HSM Adapter and ILM Controller components.

The prototype ILM-HSM adapter performs the following:

- At initialization connects to the Repository Service to retrieve the instance of the Berkeley Repository being used.
- Hooks into the Berkeley Repository allowing it to:
  - Retrieve Information Objects from the repository
  - Maintain a buffer of unevaluated information objects
  - Move Information Objects between storage levels
  - Access specific configuration details (currently Level 0 and Level 1 store locations)
- Maintains a specific amount (a threshold) of free space on level 0 store
  - Receives statistics from the File System Monitor (see below) about free space
  - Moves IOs to higher level stores to maintain a desired level of free space in the level 0 store

- React to ILM Controller commands (triggered by ILM events)
    - Free additional space
    - Change the free space threshold
- Maintains IO ID to information value pairing (used for deciding which IOs to move)

We also developed a file system monitor that is used by the ILM-HSM adapter to detect whether the amount of disk space used is beyond a threshold. The file system monitor gets the locations of the stores from the ILM-HSM adapter at initialization. Then it periodically checks the amount of free space and reports it to the ILM-HSM adapter.

The prototype ILM controller receives ILM events (implemented as Phoenix events) from the ILM Event Manager (described in last period's report), interprets the event and then carries out the proper behavior. The four ILM events that the ILM controller responds to are the following:

- *Need Space* – When the ILM controller receives a Need Space event, it calls the *move(long numBytes)* method of the ILM-HSM adapter
- *Maintain Space* – When the ILM controller receives a Maintain Space event, it sets the threshold value on the ILM-HSM adapter.
- *Info Valuation* – The ILM controller calls the ILM-HSM adapter with a request for a set of IOs, and then calls the VDF function with the set of IOs. The ILM passes the resulting VDF values to the ILM-HSM adapter.
- *Clean Up* – The actions associated with this event still need to be worked out. The strawman algorithm now is that the ILM would complete any queued tasks, then would initiate ILM-HSM movement to balance the level 0 and level 1 store (i.e., so that all the IOs in level 1 have VDF values higher than those in level 0). Then the ILM would trigger re-evaluation of all the IOs until a particular maximum time has passed or the entire store has been re-evaluated.

In the course of developing support for managing IOs associated with a mission (functionality scheduled for Spiral 1), we decided that mission relationship is just a specific example of group membership (functionality scheduled for Spiral 2). That is, a set of IOs associated with a particular mission is a "group" of IOs, similar to all IOs of a particular type, from a particular publisher, with imagery of a particular area, or any other grouping relationship. Therefore, we began designing and prototyping a grouping capability for VFILM. A group is defined as a predicate that can be matched to a set of IOs. For example,

```
publisherID="241895" AND publishTime > 02:00 AND publishTime < 0:400
```

defines a group of IOs published from a specific publisher during a two hour window of time. We developed a Group Manager that maps the predicates defining groups to fuzzy logic variables, e.g.,

publisherID="241895" AND publishTime > 02:00 AND publishTime < 0:400 $\Rightarrow$

MissionState = ActiveMission

The group states associated with an IO can be translated by Fuzzy Variable beans into numerical values used to evaluate IOs. These numeric values are then interpreted by the fuzzy logic rules in the ILM to determine the IO's membership in various fuzzy sets. In the case of the above example, sample rules would be

MissionBean:

```
          if( stateList.contains("ActiveMission")
               missionRelated = 1.0;
          else if( stateList.contains("FutureMission")
               missionRelated = 0.5;
          else missionRelated = 0.0;
```

FCL snippet:

```
          FUZZIFY missionRelated
                    TERM high := (0.3, 0) (1.0, 1)
                    TERM low := (0, 0) (0.3, 1.0)
          ENDFUZZIFY
          …
          IF missionRelated IS high THEN relevance IS high
          IF missionRelated IS low THEN relevance IS low
```

We plan to have a richer set of examples in the upcoming demonstration.

We began defining a scenario for demonstrating the VFILM prototype. The demonstration would involve multiple missions (with the associated mission prep, start, and end events), and with IOs of various sizes. The ILM would attempt to maintain a particular threshold of free space, and would need to manage the free space as missions start and end, including overlap, and as mission prep events occur. The demonstration would show the ILM ability to control movement of information, would show the values indicating how information value depreciates as the mission proceeds, and would show the impact on running queries of the ILM processes.

We also started developing an experimentation plan during the report period. As of the time of this report, we have defined experiments for four of the twelve defined VFILM metrics.

April 2010

During the report period, we continued working on the Spiral 1 VFILM prototype, shown in Figure . We integrated, tested, and refined the prototype elements designed and developed during previous report periods, and designed and developed two new elements of functionality.

The first element of new functionality that we designed and developed was support for mission events and groups. As reported last period, we began implementing IO grouping as a superset of the association of IOs with missions, i.e., treating mission association as an example of more general group membership. During this report period, we designed and developed an ILM Indexing component that we use to associate IOs with groups. The prototype Indexing component does the following:

- Maintains an instance of Berkeley XML DB, containing the Context ID of each IO, as well as the fields over which group predicates reference.
- Provides a way of retrieving all IOs associated with a group, specifically for the purpose of performing information valuation in response to events associated with a certain group. For example, a mission end event can affect the valuation of the group of IOs relevant with that mission.
- Returns the group states associated with each IO in a given list. These are used by the Value Depreciation Function to determine group state when an IO is evaluated.

- Inserts new IOs into the Index.

As reported during the last report period, we have identified four primary ILM events that the initial ILM prototype needs to respond to. Mission events (such as the preparation for a mission, mission commencement, or mission completion) and System Events (such as disk full events) are mapped to ILM events by the ILM Event Manager, and the ILM events then are interpreted by the ILM controller and can result in IO evaluation, triggering information valuation, information movement, or both. The four ILM events that we have implemented the prototype ILM controller to respond to are the following: *Need Space*, *Maintain Space*, *Info Valuation*, and *Clean Up*. As the other three were implemented during the previous report period, the other piece of new functionality that we concentrated on during this report period is the ILM controller's *Clean Up* function. We designed an initial version of the *Clean up* behavior that consists of sorting IOs stored in different levels of storage so that higher valued IOs are on lower level stores while also maintaining space thresholds on all levels of storage. We began prototyping an implementation of this functionality in the ILM controller.

Our integration and refinement of the other VFILM components have brought us to a demonstrable state with the Spiral 1 VFILM prototype. During the report period, we successfully tested the following scenario:

1. Mission client sends to the Event Notification Service a *GroupEvent* containing a predicate describing a group, and a state of "ActiveMission".
2. Group manager receives the *GroupEven*t from the Event Notification Service, and updates its internal map of group states.
3. A publisher client pushes two sets of Information Objects. One set is in the group set to "ActiveMission", the other is not.
4. As the IOs are archived, the HSM Adapter passes them to the Value Depreciation Function for evaluation. The VDF uses a set of rules based on IO size and Mission Status. IO's in the ActiveMission group are ranked more valuable.
5. After enough IOs are published a File System Monitor that was running in the background triggers a *Move* event.
6. Some of the IOs not in the "ActiveMission" group are automatically moved to a higher level store to bring the amount of free space in the Level 0 store to the acceptable level.

During this period we also started documentation necessary for a user manual. We also refined the demonstration scenario that we are targeting for the upcoming technical interchange meeting with AFRL. Finally, we refined the draft experimentation plan. As of the time of this report, we have defined draft experiments for nine of twelve defined VFILM metrics.

May 2010

During the report period, we continued working on the VFILM software and completed development of the Spiral 1 prototype. We also developed the support code, including clients, scripts, and GUIs, to support a demonstration of the prototype. We also conducted a technical interchange meeting with AFRL and continued to make progress on an experimentation plan. Details on each of these follow.

During the report period, we completed development of the Spiral 1 prototype *Information Lifecycle Management* service, shown in Figure . The prototype consists of the following elements:

- An ILM Event Manager that listens for incoming mission/system events and translates them into internal ILM events.
- An ILM Controller that reacts to ILM events, triggers IO evaluations, and triggers HSM actions
- The Value Depreciation Function that evaluates information objects
- The ILM-HSM Adapter that abstracts away the specifics of the HSM and Phoenix Repositories being used.
- The File System Monitor needed to support ILM-HSM Adapter operations for maintaining thresholds
- The Group Manager that maintains an index (the *ILM Index*) of predicates defining groups and identifiers for the groups.

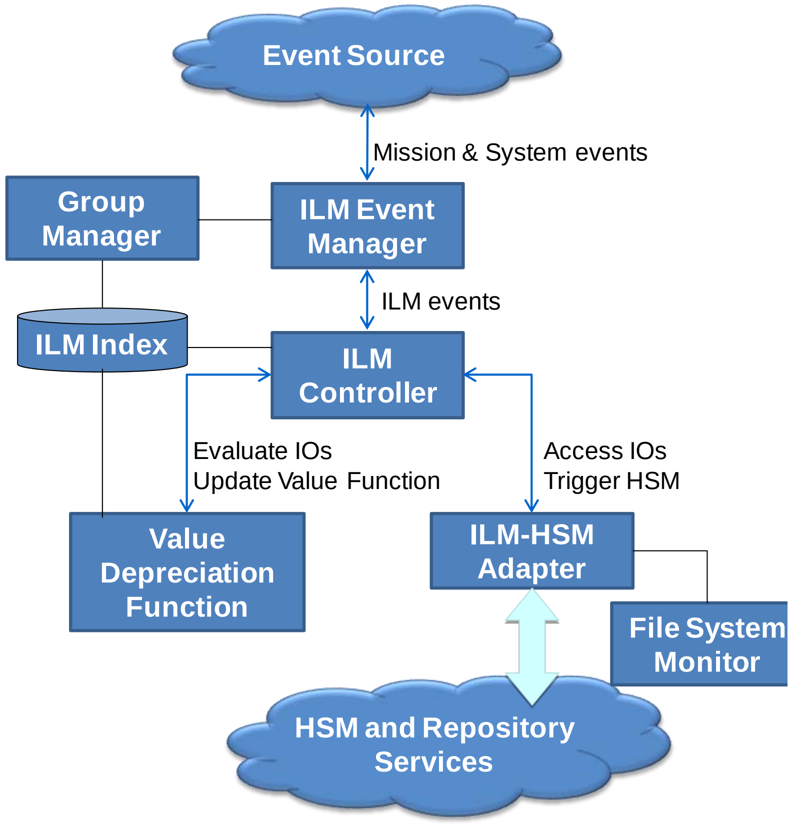


Figure A-3. Design of the Spiral 1 ILM Service as of May 2010.

The prototype passes mission events (including *mission prep*, *mission begin*, and *mission end*) events through a Phoenix Event Channel to an adapter that maps them to ILM events (serving as our first prototype *Mission Model*). The ILM Event Manager then passes the ILM events to the ILM Controller, which acts on the ILM events by running the Value Depreciation Function over sets of IOs, requesting the ILM-HSM Adapter to move IOs, or both.

The ILM controller responds to ILM events as follows:

- *Need Space* – Calls the *move(long numBytes)* method of the ILM-HSM adapter
- *Maintain Space* – Sets the threshold value on the ILM-HSM adapter
- *Info Valuation* – Calls the ILM-HSM adapter with a request for a set of IOs; calls the VDF function with the set of IOs; and passes the resulting VDF values to the ILM-HSM adapter.
- *Clean Up* – Sort the IOs in the different levels of storage so that higher valued IOs are on lower level stores, while maintaining space thresholds on all levels of storage.

The VDF is implemented using jFuzzyLogic, as described in previous reports.

The ILM-HSM Adapter utilizes the existing Phoenix Repository Service to retrieve the instance of the Berkeley Repository being used. It then interfaces to the Berkeley Repository to retrieve IOs, maintain a buffer of unevaluated information objects, move IOs between storage levels, and access the location of the various storage levels. The ILM-HSM Adapter also responds to ILM Controller commands, including to free up space or to change the *free space threshold*.

The free space threshold is used by the ILM-HSM Adapter to maintain a desired threshold of free space on the Level 0 store. To accomplish this, we developed a *File System Monitor* that provides statistics periodically about the amount of free space in Level 0. When the File System Monitor indicates that the amount of free space has crossed the threshold, the ILM-HSM Adapter moves IOs to Level 1 store to restore the desired level of free space in the level 0 store.

Finally, the ILM-HSM Adapter maintains IO ID to information value pairing (used for deciding which IOs to move).

We attended a TIM at AFRL on May 26, 2010, where we presented on the status of the VFILM project, including the following:

- The goals of the VFILM project.
- The schedule and progress since the last review.
- The design, implementation, and status of the Spiral 1 VFILM prototype.
- The VFILM experimentation plan.
- The next steps.

We also demonstrated the Spiral 1 prototype at the TIM. In preparation for this, during the report period we developed specific fuzzy sets and FCL rules defining a prototype VDF, created publishing and querying Phoenix clients, created a graphical user interface, and developed a demonstration scenario and scripts.

For the demonstration of the current prototype, we defined three fuzzy sets, representing Mission Association, Age, and Size. The membership functions defining the IO Size set (*ioSize*) and the Mission Status set (*missionStatus*) use piecewise linear functions, while the membership function defining the IO age (*age*) uses a sigmoidal function. We developed Java classes to extract the actual age, size, and mission status values from an IO and a mapping of these classes to the corresponding FCL variables.

We then developed a set of FCL rules combining these fuzzy variables into membership in a *move* output set, representing the relative depreciation value of a given IO. The FCL rules assign weighting to the variables, with *missionStatus* weighted the heaviest, then *age*, and finally *ioSize*. Membership in the output set *move* represents the partial order used by the ILM and ILM-HSM adapter to choose the IOs to move when IOs need to be moved.

The demonstration showed three dynamic and overlapping missions, and demonstrated IO valuation based on mission events, movement of IOs between storage levels, IO valuation and movement based on a system event (the amount of free space in Level 0 crossing a threshold), and the *Cleanup* action to balance the amount and value of IOs between levels of storage.

During the demonstration, we showed the display in Figure that indicated the number of IOs (Y axis) in each level of storage using colors (red for Level 0, blue for Level 1) and the VDF valuation (i.e., level of membership in the *move* set) shown on the X-axis.

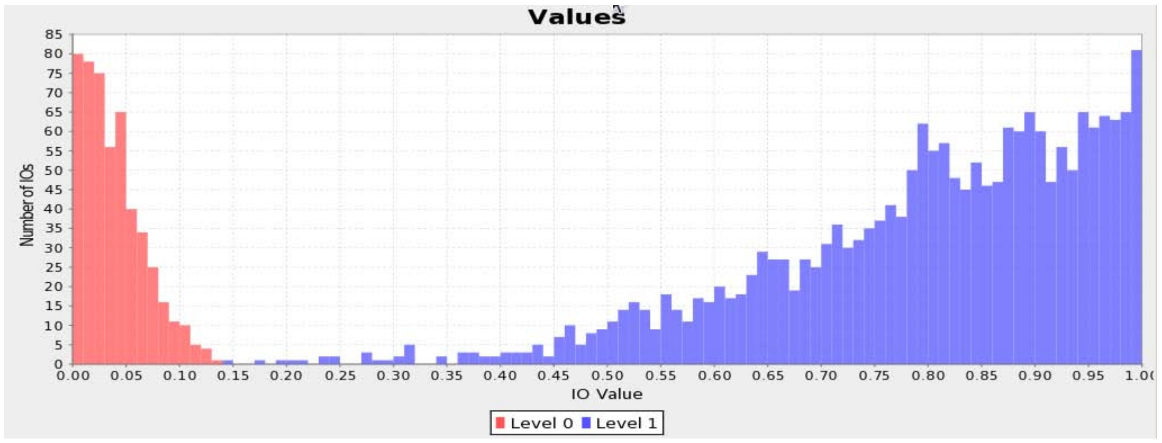


Figure A-4. VDF valuation distribution display from the May 2010 Spiral 1 demonstration.

We also showed the display in Figure that indicated the amount of free space in Level 0 and the threshold that would trigger a system event to move IOs. The X axis is a rolling window of time as the demonstration proceeded and the Y axis indicates the free space in MB.

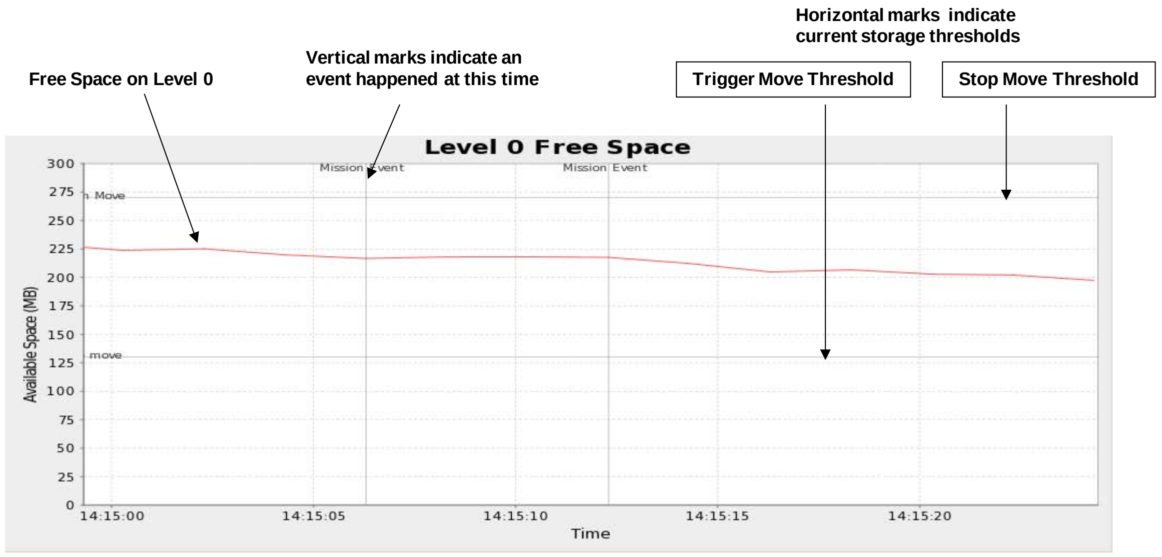


Figure A-5. Free space and threshold display from the May 2010 Spiral 1 demonstration.

During this report period, we completed a first draft of an experiment plan and began reviewing it internally. We plan to provide it to AFRL during the next report period.

During this period we also continued writing a user manual. We also plan to provide that to AFRL during the next report period.

June 2010

During the report period, we wrapped up Spiral 1 efforts and commenced Spiral 2 efforts. We developed and delivered documentation for the Spiral 1 prototype and an experimentation plan. We also commenced development of the Spiral 2 prototype in parallel with conducting experiments on the Spiral 1 prototype. Details on each of these follow.

We wrapped up Spiral 1 efforts with the production of documentation of the prototype software in a Spiral 1 *VFILM Installation, Operations, Administration, & Demonstration Guide*, and delivery of the document to AFRL for Government comments on June 10, 2010. The document describes how to install the VFILM Spiral 1 prototype software, how to build and configure the VFILM Spiral 1 prototype, and how to run the VFILM Spiral 1 Prototype Demonstration, which was shown at the VFILM TIM on May 26.

Also during the report period, we commenced Spiral 2 development efforts. We began the design and implementation of a policy system for the Information Lifecycle Manager service. Information grouping, storage thresholds, task priority, and the triggering of ILM events need to be controlled by policy. Our initial design calls for a policy input interface that allows multiple sources to manage ILM policies, which are based on the policy definition used in the Quality of Service Enabled Dissemination project. The ILM Event Manager is the primary source of policy inputs for the current prototype. This allows policy changes to be passed through the system as standard Phoenix Events. It is also possible to create an ISQM Listener that implements the policy input interface, allowing the ILM to take advantage of QED policies when appropriate (for instance inheriting rules related to Query operations). Additionally we plan to modify our current group definitions to better match the policy definitions from QED (specifically with respect to importance and precedence). This will add flexibility to the ILM group definitions, while also increasing compatibility between the two systems.

We also developed a persistent store for IO values, i.e., the results of VDF valuation. Previously, we were using in memory data structures to store the value of IOs, which required all IOs stored in the archive to be reevaluated every time the service started. We implemented code to use a Berkeley DB to store the results of IO VDF valuation, so the values are now persistent from run to run. As part of this, we developed a *Value Store Interface*, which allows for easily replacing the new Berkeley DB Value Store with the previously used In Memory Value Store, or any possible future implementations.

We also made tuning and bug fix improvements to the VFILM prototype, several of which were motivated by observations from or bugs discovered during experimentation (described below). These included changes to the ILM service and its constituent components: the Event Manager, Controller, Group Manager, ILM Index, and ILM-HSM Adapter.

Also during the report period, we delivered a draft *VFILM Experiment Plan*, Version 1.0 to AFRL for comments. We also began conducting experiments as defined in the plan. As part of setting up the experiments, we revised some of the experiment definitions to reflect more accurately the current state of the VFILM design and prototype. We plan to revise the document and provide a revision to AFRL that accurately reflects the experiments that are conducted.

We conducted Experiment 1 – ILM Responsive to Events, and confirmed the hypothesis and sub-hypotheses. We also conducted Experiment 2 – Maintain Level 0 Store, and confirmed its hypothesis and sub-hypotheses. For each hypothesis, we have written a JUnit test that can be run at build time to ensure that the prototype is functioning properly. We also wrote supporting code in the form of several wrapper classes that allow us to time relevant operations without making source changes that must be reversed later.

We developed much of the supporting code needed to run the Scalabability (Experiments 3 and 4), Correctness compared to Baseline Phoenix (Experiment 5), Performance compared to Baseline Phoenix (Experiment 6), and Mission Relevance (Experiment 7). There is some additional setup work to be done before some of these experiments can be run, e.g., scalability testing requires long-running operations that must be repeatable so we need scripts to automate the experiment runs. We will be completing this support software and experiments in the coming months.

July 2010

During the report period, we continued Spiral 2 efforts. We refined several of the experiments defined in the *Experiment Plan* document and executed all of the proposed experiments. Additionally we created a draft of the *Experiment Results* document encompassing our results for six of the seven defined experiments. We also refined the design of and implemented policy capabilities for the VFILM Spiral 2 prototype. Details on each of these follow.

We revised several sections of the *VFILM Experiment Plan.* These changes mostly clarified experimental procedures and the metrics being used to evaluate results. The only significant change had to do with Experiment 7 *Mission Effectiveness.* We felt that the previous formulation did not allow for a straightforward, objective conclusion. We have narrowed its scope to focus on the cost of the added flexibility provided by our use of Fuzzy Control Logic. The VDF's flexibility and potential *"Mission Relevance"* will instead be treated qualitatively in the form of a demonstration.

As reported in the previous period's status report our initial experiments confirmed the hypothesis and sub-hypotheses for both Experiment 1 – ILM Responsiveness to Events and Experiment 2 – Maintain Level 0 Store. During the last reporting period we completed all necessary supporting code for experimentation and executed the remaining experiments. Our tests confirmed the hypothesis and sub-hypotheses for Experiment 3 – Scalability of the VDF. The two sub-hypotheses for Experiment 4 – HSM Scalability (time to move IOs scales linearly with the number of IOs moved and time to move IOs scales linearly with the number of bytes moved) were confirmed if all Information Objects were assumed to be the same size, the hypotheses were refuted in the more general case. The results of Experiment 5 – Correctness, confirmed our hypotheses that query and publish actions would behave the same way in the VFILM prototype as compared to Baseline Phoenix. Similarly, the results of Experiment 6 – Performance, confirmed our hypotheses that the performance of query and publish actions in Baseline Phoenix would be largely the same as the performance of query and publish actions in the VFILM prototype. Lastly the results of Experiment 7 – Mission Effectiveness, indicate that the cost of using a Fuzzy Control Logic based valuation function is not prohibitively higher than the cost of some simple function, specifically IO age.

Also during the reporting period we drafted an initial version of the *Experiment Results* document. This document currently contains detailed results and explanations of Experiments 1 through Experiment 6, and will be amended to include Experiment 7. Additionally, we have started documenting the process of running the software for each experiment; a full write up of which will be included in our *VFILM Installation, Operations, Administration, & Demonstration Guide.*

We also refined the design of and implemented policy capabilities for the VFILM prototype. Policy changes are carried out by ILM events created by objects implementing the Event Handler Interface inside the ILM Event Manager. The Event Manager passes incoming Phoenix events to the appropriate event handler which then creates the appropriate ILM events. This allows Phoenix events to trigger policy changes. The Mission Domain Model is an example of one such event handler. Mission Events are processed by the Mission Domain Model, which then triggers appropriate policy changes (and other actions) inside the ILM. Phoenix events are not the sole way of triggering an event handler. For example, an ISQM Listener Event Handler could communicate directly with the ISQM and trigger changes in the ILM policy as needed.

During the reporting period we also began to outline a paper showcasing VFILM for submission to SPIE Defense, Security, and Sensing 2011. The abstract for the paper is due 10/11/2010 with the actual manuscript due 2/14/2011, allowing us the time to receive approval before publication as specified in the contract.

We also continued to make tuning and bug fix improvements to the VFILM prototype, several of which were motivated by observations from or bugs discovered during experimentation. These included changes to the ILM service and its constituent components: the Event Manager, Controller, Group Manager, ILM Index, and ILM-HSM Adapter.

August 2010

During the report period, we continued Spiral 2 efforts. We finalized and delivered the Experimentation Plan document as well as the Experimental Results document. We also implemented three new event handlers which have allowed us to streamline the design of the ILM components, expand our policy capabilities to be compatible with QED, and demonstrate the flexibility the ILM service can provide. Additionally, we have refined the design of and implemented policy capabilities for the VFILM Spiral 2 prototype. We also have started work on a demonstration for the technical interchange meeting in September. Details on each of these follow.

We revised and delivered the *VFILM Experimentation Plan;* the plan had already been through several revisions and only minor edits remained. Also during the reporting period we revised and delivered the *Experiment Results* document. Along with minor edits and corrections we also amended the draft from the previous reporting period to include the results from Experiment 7. Additionally, we have finished documenting the process of running the software for each experiment; the full write up of which will be included in our *VFILM Installation, Operations, Administration, & Demonstration Guide.*

We developed three additional implementations of the Event Handler Interface. The *Location Manager* subscribes to track data related to a certain unit and triggers group and valuation events to highly value IOs that are located in the vicinity of the tracked unit. The *File System Monitor*, which was previously embedded inside of the ILM-HSM Adapter, has been refactored into an event handler that triggers movements when free space in level 0 drops below a certain threshold. Lastly, the *ISQM Listener* implements the *Policy Change Listener* interface defined in QED. When an ISQM service triggers a QED policy update the ISQM Listener translates the QED policy into an ILM Group allowing for QED policies to be used by the VFILM prototype for IO valuation. These three event handlers both demonstrate the flexibility of the VFILM prototype and expand upon our previously existing policy capabilities.

We designed and implemented the bulk of the code necessary to demonstrate the Spiral 2 VFILM prototype during the upcoming technical interchange meeting. This includes a new GUI that displays a Cartesian grid with marks representing the storage location and geographic location (as determined by metadata) of information objects. Additionally, we developed a mock ISQM service that implements the same interfaces as the ISQM service in QED and functions in a similar, albeit simpler, manner. The interaction between the ISQM Listener and the mock

ISQM is the same as we expect between the ISQM Listener and an actual ISQM service, the simplicity has simply been introduced on the opposing side of the ISQM service which would interact with other QED components.

During the reporting period we also added the use of a thread pool for parallel operations to the Value Depreciation Function and the ILM Controller. This allows us to not only improve performance, but also for potential integration with QoS resource management. We also continued to make tuning and bug fix improvements to the VFILM prototype several of which were motivated by observations from or bugs discovered during experimentation. These included changes to the ILM service and its constituent components: the Event Manager, Controller, Group Manager and ILM-HSM Adapter.

September 2010

During the report period, we continued Spiral 2 efforts. We designed and began implementing a scheme to manage the movement of metadata and how queries interact with metadata and payloads moved to backing stores. Related to this we also modified the design of the VFILM prototype to be compatible with a Repository Service that is managing multiple repositories. We also hosted a technical interchange meeting with AFRL at BBN where we demonstrated the Spiral 2 prototype. Additionally we have drafted the abstract of a paper showcasing VFILM for submission to the SPIE conference on Defense, Sensing, and Security.

During the reporting period, we finished the code necessary to demonstrate the Spiral 2 VFILM prototype. This included some modifications to our Mock ISQM service, along with several publishing clients to carry out the desired scenario. The scenario included three units publishing track and image data as they traversed an area taking part in three different missions. The demonstration was carried out during the technical interchange meeting on September 9th 2010.

During the reporting period we designed and began implementing the ability to move metadata. This movement includes not only the IO metadata located in the repository, but also data in the ILM Value Store, and the ILM Index related to that IO. To minimally disrupt the current Phoenix Repository implementations we have opted to move the data from one Repository, ILM Index, and ILM Value Store, to a second set located elsewhere (i.e., on a different level of storage). We modified the VFILM prototype so that it works with multiple repositories. This is something the Repository Service has supported but was not previously addressed by the ILM Service. These changes bring the ILM Service more in line with the Phoenix Repository Service. As a part of this, we streamlined the configuration of the VFILM prototype so that the ILM HSM Adapter now extracts the necessary settings from the Repositories being used, allowing the ILM Service to be dropped in alongside a Repository Service with minimal configuration ahead of time.

We also started work on how to manage queries which are executed over multiple storage levels. We created an ILM Query Context which extends the standard Phoenix Query Context. The ILM Query Context contains a range of levels to query over and a range of levels from which to return results. It is important to note that the ILM Query Context will act as a normal Query Context class if the query is sent to a Repository that is not ILM Compatible, and if a standard Query Context is sent to an ILM Compatible Repository it will execute and return results spanning all levels. This allows us to achieve the desired functionality and remain compatible with baseline Phoenix implementations.

During the reporting period we also drafted the abstract of a paper showcasing VFILM for submission to the SPIE conference on Defense, Security and Sensing.

We hosted a Technical Interchange Meeting (TIM) with AFRL at BBN on September 9, 2010. At this TIM, we presented on the current status of the VFILM project, including the following:

- The VFILM project goals, schedule, and progress since the previous TIM (in May 2010).
- The design and prototype implementation of VFILM.
- VFILM experimentation and results.
- Demonstration of the current VFILM prototype.
- Summary and next steps.